Detección de Intrusos Basada en Host (HIDS) - OSSEC

Host-based intrusion detection system (HIDS) - OSSEC

Trabajo de Fin de Grado

Adrián Ruiz Householder
Daniel Candil Vizcaíno
Guillermo Sánchez-Mariscal González

Director: Marcos Sánchez-Élez Martín

Grado en Ingeniería Informática
Universidad Complutense de Madrid - Facultad de Informática

Curso académico 2019 – 2020

Índice

Resumen

En este trabajo se ha estudiado y mejorado un detector de intrusos en la red, basado en comportamientos extraños en el host (Host Intrusion Detection System - HIDS). Este tipo de herramientas de monitorización, buscan detectar anomalías en el comportamiento de la máquina anfitrión que puedan indicar un riesgo potencial, para lo que se crean y revisan los logs del sistema.
Durante este trabajo hemos estudiado diferentes HIDS llegando a la conclusión de que el sistema OSSEC cumplía todos los requisitos que buscábamos para este proyecto. Esto nos ha llevado a que gran parte del trabajo realizado haya consistido en realizar un estudio pormenorizado a nivel de código y de funcionalidad de todas las posibilidades de monitorización que ofrece esta herramienta. Una vez comprendido el funcionamiento interno hemos rediseñado la herramienta de forma que cualquier usuario que actualmente trabaje con dashboards similares, pueda fácilmente dar uso a todas las funcionalidades de OSSEC. Esto nos ha llevado a generar todo un nuevo frontend y modificar partes del código del backend para que la nueva representación gráfica fuera eficiente. Podemos decir que como resultado hemos conseguido que una herramienta potente, pueda volver a ser considerada novedosa, tan solo atendiendo a las mejoras de la apariencia de la aplicación.

Keywords: HIDS, Sistemas Operativos, Seguridad, Bootstrap, Frontend, Monitorización

Abstract

Keywords: HIDS, Operating Systems, Security, Bootstrap, Frontend, Monitoring

1. Introduction (EN)

An IDS (Intrusion Detection System) is a system which identifies, monitors and responds towards suspicious activities while two or more machines are communicating. It gathers data from the system, and when evidence is detected, a response protocol starts. An HIDS is a host-based IDS, that analyzes incoming and outcoming data packets from a specific device, notifying the system administrator if any anomaly is detected. The HIDS analyzes and audits the data that the OS and the applications have gathered. We will now discuss how intrusion detection works according to levels, the difference between IDS and the future trend of these tools

1.1 OS Level Intrusion Detection

The collected data originates from file modifications, logs, core operations, system calls... The nature of the data makes it difficult to handle unless it comes directly from the core of the OS.

1.1.1 Audit Data Recollection

It is essential that the data that has been gathered has not been modified or exposed or made vulnerable. The one in charge of the integrity of the data is the OS. However, the OS's primary function is not detecting intruders: it is the IDS who accesses the OS's data and actually uses it.

Each OS is equipped with their own audit mechanisms. SOLARIS, for example, is based off the BSM (Basic Security Module), forcing each system call to be registered in a log, with the corresponding user's information, allowing a root user to modify the functionality.

Linux comes with SNARE and LIDS, working at Kernel level as well:

- **SNARE**: Accesses the system call logs. It has a pattern-recognition module, as well as a visual tool that filters the gathered data.

- **LIDS**: It makes accesses to the standard control layer easier, and doesn't work as an IDS as is. It is also useful for data recollection, but it mostly monitors file accesses. Being root does grant the user every privilege, so it doesn't compromise the integrity of the data. There are two working modes:
    - CAP_LINUX_IMMUTABLE: protects files upon modification.
    - CAP_NET_ADMIN: protects network configuration files.

There's a specific OS log that has severe problems such as not saving formatted data, or saving it in a randomly-formatted manner. This makes automatic study and analysis of the data very difficult.

Microsoft Windows's audit system is divided in three events:

- System Log (SYSEVENT.EVT): Contains Windows service and driver data, as well as system startup and error events. In a network environment, it also generates browser events.
- Security Log (SECEVENT.EVT): It records system boots and shutdowns, privilege modifications and user logins and logouts.
- Application Log (APPEVENT.EVT): Application-generated events.

All three events can be monitored through tools such as the Windows Event Viewer, and they are accessible via the Windows32 API.

1.1.2 Bad Use Centered Approaches

The detection systems store specific attack descriptions, written in its own language. STATL [1] is a specific example of such a language that interprets an attack as a state machine with transitions. Due to the fact that, these attacks are usually very complex, a state machine would not suffice (the size would be unmanageable), therefore the data is stored in multiple variables.

The event space has a filter that depends on the mode the application is running on. Since multiple instances of the same attack can raise at the same time, STATL is equipped with several prototype case-scenarios which already have stored attacks in them. The evolution of the scene is defined by its transitions and its associated actions, which describe the events that trigger the transitions.

These scenario-prototypes are described by STATLS operation semantics. The prototypes define a global environment, and the instances represent individual attacks currently activated. The evolution of the transitions depends on the type of the transition, which can be consuming, non-consuming and unwinding:

- A non-consuming transaction represents a step of an activated attack and doesn't prevent future instances from arising from the root node of the transition. When a non-consuming transition is taken, the root state is kept valid, as well as the goal state.
- However, a consuming transition makes the root state become invalid, changing the state of the attack.
- An unwinding transition allows roll-back to previous states, which allows overcoming deadlocks in case an action, for example, the deletion of a file, makes the event unable to progress.

An example would be an attack that starts with the insertion of a malicious .rhost file into the home root directory via FTP, allowing the attacker to access the directory remotely without the need to login or enter any password.

1.1.3 Anomaly Based Approaches

Similarly to the previous approach, this detection is based on the creation of models of "normal" user behaviour, called "profiles". This procedure allows the detection of numerous unknown attacks, but it also generates a large amount of false-positives. Data extracted from a normal use of a system is gathered and then compared to an inadequate use of the same system so as to find anomalies.

A model is a set of procedures that are used to evaluate a number of characteristics of a system call. It can operate by either learning or detection mode. In learning mode, the model is trained by the normal use of the system, gathering values considered to be "normal" in the execution of a program. Detection mode on the other hand, consists on returning the probability that a system call will generate a valid result. When, for example, getting the size of a string: the standard input is usually composed of one hundred characters, the majority of them being human legible (commands). In general, system calls are represented by canonical names of previously found files in the system. If there is a failed file open attempt recorded in the log, the attacker can take advantage of the event and send malicious code in the "open" event, of several hundred bytes. The objective is to reach the size of the required arguments of a system call and to detect abnormal cases, using statistics of real well-known strings. After the detection phase, the system call arguments are compared with the model and are analyzed to see if they fall under legitimate ranges.

The character distribution model has the advantage of being difficult to bypass when being attacked by well-known methods of hiding malicious code within a string. These models raise alerts when certain patterns are detected. Altering these sequences into very similar instructions would make the signature-based model to fail when trying to detect anomalies, whereas the character distribution model would detect the distorsion.

No precise definition of what a model represents can be made, since a lot of attacks are based on "normal" behaviour. The attacks are called "mimicry attacks" and arise when there are design flaws in the system or if the quality of the input-triggered events is poor. The attacker is usually detected since he is not aware of such intrusion detection systems, and if he didn't know, he would be forced to behave as a "normal" user.

1.1.4 Specification Based Approaches

Anomaly detection systems are based on the analysis of the static data from the application code. The models are created for and by the application itself.

The techniques used are less prone to raise false alarms, since they follow strict and complete policies, which make the system work considerably well. Nonetheless, generating such a complex policy for applications and scenarios is not trivial.

1.2 Application Level Intrusion Detection

A fundamental source of audit data for an HIDS is the data that applications generate. This data is more extensive and reliable, making it easier to detect which program is responsible for generating a specific event. Unfortunately, the data is very specific, and the HIDS needs to treat each case differently depending on the application.

1.2.1 Audit Data Recollection

The majority of OS use shared, centralized log files, between the OS and the audit applications. The data handled by the audit applications can also be found in specific log files within the application directories. Logs which contain error entries are usually more relevant, since the errors usually occur when malicious behaviour is underway.

Although log files have very detailed information, since every application writes entries in a different format, the HIDS needs to adapt to the application individually. Moreover these logs only gather information after the operation has been done (and with little detail), which doesn't allow the system to act in a preempting manner. In some applications, it is possible to modify the log files in order to increment the verbosity of the data, but even so, the OS might not always have enough information to interpret the triggering operation, and such files usually occupy a lot of memory.

In order to overcome some of these limitations, systems which are integrated inside the application, who analyze the data at the same time that the application is executing have been developed, allowing a more preemptive approach. The closer the IDS is integrated to the application, more information can be gathered, hence reducing the amount of false alarms. So as to effectively integrate these systems, the application must have an interface in the form of an API, or subroutine call responses, or simply extend the application if it's open source.

An example of an integrated IDS would be the "Honeypot", which willingly attracts attackers by allowing them to easily see vulnerabilities. They are best used for learning how attackers behave, as well as to distract them from their initial objective. Depending on the liberty the attacker is given to attack the system, we can distinguish two sorts of "Honeypots":

- Low-interaction: they are fairly easy to maintain, but show little resemblance to an actual OS, hence the attacker will quickly see that they've been deceived.
- High-interaction: these systems on the other hand try to completely mimic the behaviour of a fully equipped OS, allowing the IDS to collect a lot of information from the attacker, since no specific behaviour is expected. As expected, these systems are more tedious to maintain.

1.2.3 Bad Usage Approaches

Swatch is a simple signature-based system made for application-data auditing in UNIX. It monitors system-generated messages while they're being written via the syslog file, as well as warn the system administrator about possible errors.

Other similar systems, like LogSentry, extend the surveillance tasks to another file system, like the ones created by system daemons, TCP packet wrappers... LogStat uses STATL (as mentioned in 2.2) and uses a technique based on the analysis of the log data generated after state transitions in the syslog file. WebSTAT, related with LogStat, extends the analysis of the network session logs generated by Apache.

An interesting WebSTAT attack scenario is cookie stealing. The scenario starts by registering the cookie to the start of a session by a remote client, mapping the cookie to an IP address. An idle timer is activated in such a way that, a later use of the cookie by the same user will reset it. When the cookie expires, the mapping will be undone. However, if the client used a cookie mapped to another IP linked to another client's session, this will result into an attack, raising an alarm.

Some attacks act in multiple sub-steps, making them seem harmless. A single use of a cookie is usually considered normal, and unless two clients share the same cookie, no malicious behaviour will be recorded.

1.2.4 Anomaly Based Approaches

An application-based anomaly detection system generates a behaviour profile based on the expected activity a client will do while using the application. A profile can be a collection of all the use cases of the program.

Examples of systems:

- DIDAFIT (Detecting Intrusions in Databases through Fingerprinting Transactions) [2]: This system records access patterns to a database with legitimate fingerprint authentication, allowing it to identify unwanted intrusions.
- Another example would be a system which monitors the normal behaviour of the clients of a web application. Given the client-side petitions, the system generates models for bulk petitions with different characteristics.

A change in these access patterns indicates an attack. When the expected traffic of an application suddenly raises, it could either mean that an attacker is trying to find vulnerabilities or trying gain access by brute force. If the attacker is knowledgeable enough, he will mostly try to reduce the access frequency, so as to not raise awareness.

The order in which programs are accessed is also a good example of a pattern. Web applications are usually made out of server-side programs, that together, make up the whole functionality of the application. The order in which these programs are accessed can identify if the system is under an attack.

1.2.5 Specification Based Approaches

The system-call interface is usually in charge of containing the behaviour specification of the application, but a much more robust method of specifying it is done by defining the input and output data exchanged with the users.

VDM [3], a specification language that focuses on DNS, defines the security objectives of a service: a server should solely use consistent DNS data with the corresponding domain name. Based on this specification, in order to comply with the security standards we want to achieve, a DNS wrapper is implemented, which examines incoming and outgoing traffic. In cooperation with the authoritative name server, messages that go against the security policy are detected, and discarded. Some examples of detected unwanted messages would be spoofing and cache poisoning. This can be extended so as to avoid calls to library functions, argument modifications and return data alteration.

1.3 Related Techniques

Integrity checkers are tools that allow us to determine if a file has been tampered with. Even though they don't form part of HIDS, they're used to detect malicious behaviour. Tripwire is one of the most known detectors, supervising the values of a file that must never be changed. The tool stores the data hash and uses it in order to detect if a file has been modified.

Unfortunately, one of the disadvantages of these integrity-check tools is that they can only work in a reliable manner when offline, within a non-compromised OS. This is due to the fact that integrity checkers base their analysis on OS routines to access file systems. If the operating system has been modified in a compromised host, these inspectors may not detect modifications in files. Once the kernel is infected, it becomes rather difficult to understand whether or not a system has been meddled with without the help of hardware extensions.

A well known habit after a successful attack is the installation of a rootkit, a set of tools that help the attacker hide within the administrators of the operating system. Rootkits that modify system files can be detected with integrity checkers, but more modern rootkits can bypass these tools by directly modifying the kernel.

HIDS were made as a response to this. One of the most widely used techniques is the detection of modified kernel modules or discs, by searching inside strings with uncommon characters, or analyzing rootkit configuration files. In order to counterattack a hijacked OS, rootkit scanners

were developed. These tools are implemented as internal kernel modules that can directly access the kernel memory. They can monitor the integrity of memory addresses without having to rely on system calls.

Another known group of tools are antivirus. A virus is designed to copy itself in different locations within the OS, without the user's consent. An antivirus scans the physical content of a file that is usually accompanied with malicious activity and detects whether ot not malicious code is injected in them.

1.4 Host and Network-based IDS

One of the advantages of HIDS is that they can access semantically rich data gathered from operations within a host, whereas a NIDS (Network-based IDS) only analyzes network traffic, which is already a tedious task, more so when the traffic is encrypted, forcing the use of external tools to decipher it. HIDS don't need to collect as much data, since the flow of activities within the OS is slower than in a network.

Another advantage of HIDS is that they're less prone to invasion attacks, since it is more difficult to synchronize the image the IDS has generated from the application than the application itself. Last but not least, an HIDS has a better chance to respond to an attack since they are sometimes easier to identify and deal with.

On the other hand, HIDS are limited. An important disadvantage is that a compromised host can result in the deactivation or altering of the audit system or even the HIDS itself. This problem is caused because of the "all-or-nothing" approach of the privilege managing in the OS. Once someone gains administer privileges, the system becomes completely modifiable. The process to detect intruders can also considerably affect the performance of the operating system as well as the applications that are running in the audited host.

Finally, the cost of a HIDS is higher than of a NIDS, since every host within a network will need to have an HIDS installed, and not every operating system will be the same.

1.5 A Look Into the Future

In the past, HIDS were never considered to be a viable solution to detect intruders due to their performance and that an affected host will result into wrong data. Furthermore, the amount of false positives makes it very difficult to form an adequate response to an attack with the risk of it affecting legitimate users within a system.

The introduction of new and efficient detection techniques and hardware-based mechanisms to guarantee the integrity of a system has had an impact and has made IDS more popular. This tendency can also be seen in firewalls, since initially, a single unit would be deployed per network, whereas nowadays, every host is equipped wit one, making them a sort of HIDS.

If the tendency continues moving forward, the next generation of HIDS will most likely integrate host-based firewalls as well as new technology, like virus scanners and integrity checkers. Newer IDS will be able to monitor several events in parallel and relate instances of evidence between different domains. The goal would be to offer effective detection and highly reliable responses at a reasonable computational cost.

Another tendency is the integration of intruder detection mechanisms inside the OS's kernel, which is currently done outside of it. This will make responses become much more efficient, blocking suspicious operations before they cause any harm. In order for this to be viable,
strict quality protocols must be followed in the development in order to comply with the user's expectancies in terms of performance and reliability.

1.6 Conclusions

In the last decade, Network-based IDS have been chosen over HIDS. Being as easy as they are to maintain and given possibility to control several instances in just one deployment of an NIDS makes them seem the best choice. The rapid increase of network traffic and secure connection protocols however may be the end of this tendency.

The quality of the audit data available at the operating system and application level, the increasing awareness of the need of security and the improvement of precise host-based techniques have all contributed to a wider acceptance of the use of such detection mechanisms.

1. Introducción (ES)

Un Sistema Detector de Intrusos, o IDS, es un sistema encargado de identificar, monitorizar y responder hacia actividades sospechosas cuando existe comunicación entre dos o más máquinas. Recolecta datos de un sistema, y cuando se detecta una evidencia, se inicia un protocolo de respuesta. Llamamos HIDS a un IDS que es ejecutado en máquinas anfitrionas (hosts), que monitoriza los paquetes entrantes y salientes únicamente del dispositivo, alertando al administrador del mismo si se ha detectado cualquier anomalía. El HIDS analiza y audita datos proporcionados por el sistema operativo o por aplicaciones. A continuación relataremos cómo funciona la detección de intrusos según los niveles, la diferencia que existen entre IDS y la tendencia futura de estas herramientas

1.1 Detección de Intrusiones a Nivel del sistema operativo

La información recolectada es extraída de modificaciones de archivos, registros de archivos (logs), operaciones de bajo nivel, llamadas de sistema... Estos datos suelen ser difíciles de manipular a no ser que se ataque directamente al núcleo del sistema operativo.

1.1.1 Recolecta de Datos Auditados

Es imprescindible que los datos que se recojan no hayan sido modificados, ni se vean expuestos o vulnerables. De esto se suele encargar la integridad del sistema operativo. La utilidad primaria del sistema operativo no es la detección de intrusos, por lo que es el IDS el encargado de acceder a los datos del sistema operativo para dar una utilidad a los datos recogidos.

Cada sistema operativo cuenta con sus propios mecanismos de auditoría. SOLARIS por ejemplo se basa en el Kernel BSM (Basic Security Module), que hace que cada llamada sea registrada en un log, con el usuario respectivo, y permite modificar el compartimento con acceso root.

El sistema operativo Linux, cuenta con SNARE y LIDS, también a nivel de Kernel:

- SNARE: Accede a los archivos de registro (log) de llamadas de sistema. Cuenta con un módulo de reconocimiento de patrones de comportamiento, además de una herramienta gráfica que filtra la información recolectada.

- LIDS: No es un IDS per se, sino que facilita el acceso a la capa de control estándar. Es también un recolector de datos, pero actúa como control de acceso a archivos. Ser usuario root no concede todos los privilegios, por lo que no compromete la integridad de la información. Contiene los siguientes modos:

- CAP_LINUX_IMMUTABLE: protege archivos de ser modificados.
- CAP_NET_ADMIN: previene modificaciones de archivos de configuración de redes.

Existe un log perteneciente al sistema operativo que cuenta con desventajas si se utiliza directamente para la extracción de información relevante, como el almacenamiento de información con un formato no especificado, o aleatorio. Por esta razón, aplicar un análisis de manera automatizada, hacer un estudio estadístico y clasificar presuntas actividades, es complicado.

El sistema operativo MICROSOFT WINDOWS tiene un sistema de auditoría que se divide en tres eventos:

- System Log (SYSEVENT.EVT): Contiene datos de servicios y controladores de windows. Además, contiene eventos de inicio de sistema y de errores. En un entorno de redes también se generan eventos de navegadores.
- Security Log (SECEVENT.EVT): Sigue los inicios y cierres de sesión, los cambios de privilegios, los inicios de sistema y el cierre del sistema.
- Application Log (APPEVENT.EVT): Son eventos generados por aplicaciones.

Los tres logs pueden ser vistos a través de herramientas como el Windows Event Viewer y accesibles mediante la API Windows32.

1.1.2 Enfoques Basados en el Mal Uso

Los sistemas de detección almacenan una serie de descripciones de ataques. Cuentan con un lenguaje propio para describir los ataques analizados. STATL [1] es un ejemplo de lenguaje que describe un ataque como una máquina de estados y transiciones. Como no es posible describir un sistema como una máquina de estados dada la gran cantidad de posibilidades, este los recoge como múltiples variables.

El espacio de eventos tiene un filtro que depende del modo en la que la aplicación se ejecuta. Cómo es posible que haya varias ocurrencias del mismo ataque a la vez, STATL cuenta con varios escenarios prototipo que tienen ataques guardados en ellos. La evolución de escenario se define por sus transiciones, las cuales cuentan con acciones asociadas, que son las que describen los eventos que causan que la transición sea tomada.

STATL cuenta con semánticas operacionales que describen instancias de prototipos de escenarios. Estos prototipos definen el entorno global, y las instancias representan ataques individuales actualmente en acción. La evolución de estas instancias dependen del tipo de transiciones, las cuales pueden ser de tipo consumidoras, no consumidoras o desembocantes:

- Una transición no consumidora es usada para representar un paso de un ataque en acción, que no previene que futuras ocurrencias de ataques surjan desde el estado raíz de la transición. Cuando una transición no consumidora es tomada, el estado raíz se mantiene válido, a la vez que el estado destino.
- En contraste, una transición de tipo consumidora hace que el estado raíz se convierta en inválido, cambiando el estado del ataque inicial.
- Una transición desembocante permite hacer vuelta atrás (roll-back) para poder volver a los estados anteriores. En mitad de un ataque, el borrado de un archivo puede invalidar la continuación del mismo, y por lo tanto puedes volver a un estado anterior, como antes de que se creara un archivo.

Un ejemplo sería un ataque en el que el atacante crea un archivo .rhost malicioso en el directorio raíz (home) a través de ftp, permitiéndole tener acceso de forma remota sin tener que loguearse con una contraseña.

1.1.3. Enfoques Basados en Anomalías

Complementario al 'enfoque basado en el mal uso', la detección se basa en modelos de comportamiento normal de usuarios, llamado "perfiles". Cualquier desviación de los perfiles establecidos se interpreta como ataques. Con este procedimiento, se detectan mayor número de ataques desconocidos, pero se genera un alto número de falsos positivos. Se recopila información de sistemas legítimos durante su uso normal, y se compara con un uso inadecuado para descubrir anomalías.

Un modelo es un conjunto de procedimientos que se usan para evaluar un número de características de una llamada al sistema (como el tamaño de una cadena de caracteres "String"). Un modelo puede operar mediante aprendizaje o detección. En el modo aprendizaje, el modelo se entrena haciendo un uso normal, recopilando valores considerados "normales" en la ejecución del programa. El otro modo consiste en devolver la probabilidad de que una llamada del sistema obtenga un resultado válido.

Un modelo ejemplo sería el tamaño de un "String": la entrada estándar suele estar compuesta de un máximo de cien caracteres, y la mayoría consiste en caracteres legibles por humanos (comandos). Si existe input malicioso, normalmente aparece en los argumentos de llamadas de sistema. Por lo general, las llamadas al sistema se representan con nombres canónicos de archivos encontrados en el propio sistema. Si, por ejemplo, un intento de apertura de un archivo falla, y esto aparece reflejado en el log de una aplicación, se puede hacer uso de esta vulnerabilidad, de forma que el atacante envía como argumento en la llamada "open", código malicioso, de un tamaño de varios cientos de bytes. El objetivo es la aproximación de las longitudes de un argumento y detectar casos que se desvíen del comportamiento normal. Para caracterizar las longitudes de Strings se utiliza la media y la varianza de las longitudes de String reales (conocidos). Después, en la fase de detección, los valores de la llamada del sistema se comparan con el establecido, observando si aparece entre los rangos legítimos.

El modelo de distribución de caracteres tiene la ventaja de que no puede ser eludida por algunos intentos conocidos para ocultar el código malicioso dentro de un String. Estos contienen normas que elevan una alarma cuando ciertas secuencias son detectadas. Un intruso puede sustituir estas secuencias por instrucciones que tienen un comportamiento similar, y esto hace que el modelo basado en firmas no pueda detectar la anomalía con facilidad, aunque el análisis de distribución de caracteres detecta la distorsión y da un alto puntuación de anomalía.

La definición de modelos es complicada, ya que muchos de los ataques suelen basarse en comportamientos "normales". Este tipo de ataques se denominan "mimicry attacks" que pueden darse por problemas de diseño o por la baja calidad de los eventos de entrada. Los atacantes suelen ser detectados porque desconocen que existe un detector de intrusos. De lo contrario, el atacante suele comportarse de una forma más "normal".

1.1.4. Enfoques Basados en la Especificación

Mientras que los sistemas de detección de anomalías basados en el aprendizaje construyen modelos mediante el control de rastros de aplicaciones, el modelo de especificación se define a priori sin necesidad de utilizar la información dinámica del programa.

Las técnicas que utilizan las especificaciones por lo general no son tan propensos a informar falsas alarmas como sus primos basados en anomalías. Es decir, dada una política completa y exacta, estos sistemas funcionan muy bien. Por desgracia, la tarea de producir una política de este tipo de aplicaciones realistas y escenarios no es trivial.

1.2 Detección de Intrusos a Nivel de Aplicación

Una fuente importante de datos de auditoría para los sistemas de detección de intrusiones basado en hosts es la información proporcionada directamente por las aplicaciones. Los datos recolectados por la auditoría de aplicaciones dan información más extensa y fiable. Por lo tanto, es fácil determinar qué programa es el responsable de un evento en particular. Desafortunadamente, los datos proporcionados por la aplicación son muy específicos, y el HIDS tiene que tratar estos de forma específica para cada aplicación.

1.2.1. Recolecta de Datos Auditados

La mayoría de los sistemas operativos utilizan archivos de log centralizados compartidos entre el sistema operativos y las aplicaciones de auditoría. Los datos de las aplicaciones de auditoría también se encuentran en registros log específicos de las aplicaciones. En concreto los logs de error tienen mayor importancia, ya que suele recoger comportamiento malicioso.

Aunque los archivos log cuentan con información altamente detallada, al tener un formato distinto por aplicación, el HIDS tiene que adaptarse a cada aplicación de manera individual, además de que el log solamente recoge información (y de manera resumida) una vez ya ha sido realizada la operación, por lo que no puede actuar de forma preventiva. En algunas aplicaciones, es posible modificar los logs para que estos recojan más información, pero no siempre cuenta el sistema operativo con la información necesaria para poder interpretar la operación que se ha realizado, y suele pesar mucho ocupando grandes cantidades de memoria.

Para deshacerse de algunas limitaciones, se han propuesto sistemas integrados dentro de las aplicaciones que analizan los datos a la vez que la aplicación se encuentra en ejecución, lo cual actúa de manera preventiva en cierto modo. Cuanto más integrado está el IDS a la aplicación, más información se puede recolectar, y además genera menor cantidad de falsas alarmas. Para poder integrar el IDS con la aplicación de manera efectiva, la aplicación debe de contar con una interfaz adecuada en forma de API, o respuestas a llamadas de subrutinas, o simplemente extender la aplicación en caso de ser una aplicación de código libre.

Un ejemplo de un sistema IDS integrado sería el de "Honeypot", que expone de manera deliberada vulnerabilidades para incitar al atacante. Los "honeypots" son útiles para analizar y estudiar el comportamiento de los atacantes, además de distraerlos del objetivo principal. Dependiendo del nivel de actividad permitido para el atacante, distinguimos dos tipos de honeypots:

- Low-interaction: Tienen la ventaja de ser simples de mantener, pero a la vez son fáciles de detectar por el acante.
- High-interaction: De forma contraria a los low-interaction honeypots, intentan emular por completo el comportamiento de un sistema operativo, lo cual les permite recolectar una gran cantidad de información para conocer las intenciones del atacante. Además, al no asumir el comportamiento del atacante, cuentan con un entorno abierto que captura toda la actividad, permitiendo que el IDS aprenda comportamientos que no conocía previamente. Sin embargo,estos tipos de honeypot son más complejos de desarrollar y de mantener.

1.2.2. Enfoques Basados en el Mal Uso:

Uno de los sistemas basados en firmas más simples que monitoriza los datos de auditoría de aplicación es Swatch. Es un programa del sistema UNIX y supervisa los mensajes a medida que se escriben en un registro a través del fichero syslog. Swatch puede controlar la información a medida que se añade al registro, y además alerta al sistema de administración sobre problemas graves que pueden ocurrir.

Existen sistemas similares como LogSentry, que se extiende a la vigilancia de otro sistema de archivos, como los producidos por demonios (Daemon) del sistema, paquetes TCP Wrapper...

Otra herramienta es LogStat que utiliza el mismo lenguaje hablado en el apartado 2.2, STATL. Éste aplica una técnica de análisis de transición de estados al contenido del archivo syslog. WebSTAT, es una herramienta relacionada con ésta, y extiende el análisis en los ficheros de inicios de sesión, en particular, los ficheros de registro del servidor web creado por Apache.
Un escenario de ataque interesante es el escenario de robo de cookies. El escenario comienza con el registro en la cookie de un inicio de sesión por un cliente remoto, mapeando la cookie a una dirección IP. Además, se configura simultáneamente un temporizador de inactividad. Un uso posterior de la cookie por el mismo cliente da como resultado un reinicio del temporizador, mientras que el vencimiento de una cookie provoca la eliminación del mapeo para esa cookie. Sin embargo, si un cliente utiliza la cookie de sesión de otro, entonces se supone que un ataque está en marcha y se activa una alarma.

Ciertos ataques sólo se manifiestan en forma de múltiples pasos, en el que cada paso individual no es intrusivo per se. En el caso de robo de cookies, el uso de una cookie por cada cliente parece benigna, y sólo mediante la detección de que dos clientes diferentes comparten una cookie, el comportamiento malicioso puede ser identificado.

1.2.3. Enfoques Basados en Anomalías:

Un sistema de detección de anomalías, basados en la aplicación, crea un perfil de comportamiento de la aplicación, basada en la actividad esperada o "normal" de la misma. Un perfil puede ser creado mediante la especificación de todas las operaciones que se permite a una aplicación llevar a cabo.

Algunos ejemplos de sistemas:

- DIDAFIT (Detecting Intrusions in Databases through Fingerprinting Transactions) [2]: Este sistema funciona mediante patrones de accesos a la base de datos mediante huella digital de forma legítima, de modo que nos permite identificar intrusiones a la base de datos.

- Otro ejemplo es un sistema que monitoriza la actividad normal de clientes en la web. El sistema analiza las distintas consultas del lado del cliente, y crea modelos para tener una gran cantidad de consultas con diferentes características.

Los cambios en los patrones de acceso pueden indicar ataques. Cuando una aplicación se accede con poca frecuencia, pero de repente tiene una gran cantidad de accesos, se puede deber a que un atacante intenta encontrar vulnerabilidades, o un ataque de fuerza bruta. En cambio, si el atacante es bueno, suele mantener una frecuencia baja de accesos para no levantar sospechas.

Otro patrón se centra en el orden en que se accede a los programas. Las aplicaciones web a menudo están compuestos de un conjunto de programas de servidor, que juntos, implementan la funcionalidad de la aplicación. En el orden en el que se hagan los accesos a las distintas partes de la aplicación pueden determinar si se puede estar sufriendo un ataque.

1.2.4. Enfoques Basados en la Especificación

La especificación del comportamiento de una aplicación suele hacerse en la interfaz de llamadas del sistema, pero se ha sugerido especificar de manera formal el comportamiento, definiendo los datos de entrada y de salida intercambiada con los usuarios.

Un ejemplo sería el lenguaje de especificación VDM [3] enfocada a los DNS (Domain Name System), que definen un objetivo de seguridad del servicio: un nombre de servidor solo debe de usar datos DNS consistentes con los datos del nombre de servidor del propio dominio. Basado en esta especificación y para imponer el objetivo de seguridad, se implementa un envoltorio DNS, el cual examina el tráfico saliente y entrante. Para detectar mensajes que violen el objetivo de seguridad, se debe de cooperar con el name server autoritativo. Cuando se detecta un mensaje fraudulento, se descarta. Este enfoque permite detectar ataques de tipo spoofing y cache poisoning. Esto se puede extender más para evitar llamadas a funciones de librerías, modificación de argumentos y ajustar los valores devueltos.

1.3 Técnicas Relacionadas

Los "*integrity checkers*" (detectores de integridad), son herramientas que nos permiten detectar si determinados archivos han sido modificados. Estas herramientas no son parte de los HIDS pero son usados normalmente para detectar estas actividades maliciosas. Unos de los detectores más conocidos es Tripwire, que supervisa los atributos clave que no deberían cambiar de ficheros relevantes. Almacenan el hash de la información obtenida en una base de datos, pudiendo así detectar si estos se han modificado.

El límite de los detectores de integridad es el hecho de que el análisis sólo se puede realizar de forma fiable en un "offline", desconectado de la red. Es decir, el sistema de ficheros tiene que ser montado por un sistema operativo no comprometido. Esto se debe a que los comprobadores de integridad se basan en las rutinas del sistema operativo para acceder al sistema de ficheros. Si el sistema operativo en sí es modificado en un host comprometido, los inspectores podrían proporcionar información incorrecta que puede conducir a la conclusión de que no se han producido modificaciones. Una vez que el Kernel está infectado, es muy difícil determinar si un sistema ha sido comprometido sin la ayuda de extensiones de hardware.

Un hábito normal después de una intrusión exitosa es la instalación de un "rootkit". Un rootkit es una colección de herramientas fácil de utilizar que ayudan al atacante a esconderse del administrador del sistema operativo. Los rootkits que modifican los ficheros del sistema pueden ser detectados por un detector de integridad, pero actualmente existen rootkits que hacen un sobrepasan los integrity checkers modificando el Kernel directamente (ej. knark o Adore).

Para detectar rootkits a nivel de Kernel, se crearon los HIDS. Las técnicas más utilizadas son la detección módulos de Kernel o discos modificados, mediante la búsqueda de cadenas de caracteres con caracteres raros, o buscando por los archivos de configuración de determinados rootkits.
Para luchar contra el problema de un sistema operativo comprometido, se crean los escáneres de rootkits. Estas herramientas se implementan como módulos del kernel con un acceso directo a la memoria del kernel. Son capaces de monitorizar la integridad de direcciones de memoria sin tener que hacer llamadas al sistema.

Otro grupo de herramientas son los antivirus. Los virus están diseñados para hacer copias de sí mismos, en distintas localizaciones, sin el consentimiento del usuario. Lo que hacen los antivirus es escanear el contenido de los ficheros que saben que están relacionados con actividades maliciosas y detectan si hay algún código malicioso en estos.

1.4 IDS Basados en Host e IDS Basados en Redes

Una ventaja de los HIDS es que pueden acceder a información semánticamente rica sobre las operaciones realizada en un host, mientras que un NIDS (Network-based Intruder Detection System) analiza el tráfico de la red. Esto es aún más evidente cuando se encripta el tráfico. En este caso, un monitor basado en la red tiene que ser equipado con el material clave necesario para descifrarlo.

La cantidad de información que tienen los HIDS suele ser más limitada, ya que la velocidad de los eventos generados por el sistema operativo y las aplicaciones es menor que la velocidad de las que se envían los paquetes de red.

Otra ventaja de los HIDS es que son menos propensos a ataques de invasión, ya que es más difícil de sincronizar la imagen que tiene el IDS de la aplicación supervisada con respecto a la propia aplicación. Finalmente, un HIDS tiene una mejor posibilidad de realizar una respuesta enfocada porque el proceso que realiza un ataque puede ser a veces más fácil de identificar y terminar.

Sin embargo, los HIDS suelen sufrir unas series de limitaciones. Una desventaja importante es que el compromiso de un host puede permitir a un atacante desactivar o alterar el sistema de auditoría o incluso desactivar el HIDS por completo. Este problema es causado por el enfoque de “todo o nada” para la gestión de privilegios que contienen los sistemas operativos. Una vez un proceso obtiene los privilegios administrativos, es capaz de cambiar cualquier cosa del sistema.

Otra desventaja es que el proceso de detección de intrusos puede afectar sustancialmente al rendimiento del sistema operativo, y de las aplicaciones que se ejecutan en el host supervisado.

Finalmente, el coste de un HIDS es superior al de un NIDS, ya que es debido a la necesidad de instalar un IDS en cada host y de la heterogeneidad de la mayoría de los entornos (diferentes sistemas operativos y diferentes versiones de los mismos).

1.5 Tendencias del Futuro

En el pasado, los HIDS no eran considerados una solución viable para detectar intrusos debido a su rendimiento y que en caso de un compromiso no pueden compartir datos exactos. Además, el número elevado de falsos positivos dificulta una respuesta adecuada a la intrusión con el riesgo de afectar a usuarios legítimos del sistema.

La introducción de nuevas técnicas eficientes de detección y mecanismos basados en hardware que garanticen la integridad del sistema ha aumentado el interés en esta clase de IDS. Esta tendencia también se ve reflejada en la tecnología firewall (cortafuegos), que ha avanzado desde uno por red, hasta colocarse uno por cada host, las cuales podrían considerarse una especie de HIDS que analizan los datos de la red.

Si las tendencias siguen esta trayectoria, la siguiente generación de HIDS seguramente integren cortafuegos basados en el host además de otras tecnologías, como escáneres de virus y comprobadores de integridad. Los futuros IDS podrán monitorizar varios eventos a la vez y relacionar evidencias que pertenecen a varios dominios. El objetivo sería proveer una detección efectiva y una respuesta altamente fehaciente a un coste de cómputo razonable.

Otra tendencia en HIDS es la integración de mecanismos de detección de intrusos dentro del propio Kernel del sistema operativo, ya que actualmente se lleva a cabo desde fuera. Esto haría que las respuestas sean más eficientes, bloqueando operaciones sospechosas antes de que causen ningún daño. Para que esto sea viable, se deben de seguir unos protocolos de calidad muy exigentes en el desarrollo para poder responder a las expectativas del usuario en cuanto a la fiabilidad y al rendimiento.

1.6 Conclusiones

Durante la última década, los sistemas de detección de intrusos basados en la red han dominado claramente sobre los sistemas basados en el host. La facilidad de mantenimiento y la posibilidad de controlar varios objetivos con una única instalación de IDS ha inclinado la balanza hacia la solución basada en red. Sin embargo, la rápida creciente de uso de red y las conexiones cifradas, puede cambiar la situación.

La calidad de los datos de auditoría que está disponible en los niveles del sistema operativo y aplicación, la creciente concienciación sobre la seguridad de los usuarios finales, y la mejora de la

precisión de las técnicas basadas en host han contribuido a una mayor aceptación de tales mecanismos de detección.

2. Motivación y objetivos

2.1 Idea principal

La seguridad es una pieza fundamental y esencial en la informática, ya que se almacenan y se distribuyen miles de datos importantes que, si caen en manos de gente que no corresponde, pueden causar daños irreparables. Por ello nos movió la idea de realizar un proyecto de esta temática. El curso pasado, durante la asignatura de *Redes y Seguridad,* pudimos ver varios tipos de software que ayudaban al usuario estar más protegido, como por ejemplo un NIDS llamado Snort. Con esta idea, se nos ocurrió crear un HIDS, ya fuese desde cero o con la base de uno ya creado, para así comprender el funcionamiento interno de estos software y ver las diferencias que pudiese tener con el NIDS estudiado. Además, intentar, en caso de disponer de tiempo suficiente, la implementación de aprendizaje automatizado (machine learning) en este HIDS.

Entender el funcionamiento de un sistema IDS y poder desarrollar uno, no solo suponía un reto, sino que además nos dotaría de formación que no habríamos logrado de otra manera, al no ser que en un futuro trabajaremos en una empresa que implemente estas herramientas.

2.2 Objetivos iniciales

Los objetivos que en principio nos marcamos son:

- Comprender el funcionamiento interno de un HIDS.
- Incorporar nuestras propias reglas.
- Introducir aprendizaje automatizado.

2.3 Dificultades

Nuestra primera idea, una vez aceptado realizar este proyecto, fue crear nuestro propio HIDS. Realizamos una investigación pero no conocíamos ningún sistema operativo con tanta profundidad, como para poder desarrollar el HIDS desde cero, o al menos, con nuestro conocimiento. El conocimiento que tenemos de C es limitado y no disponemos del tiempo suficiente como para desarrollar la aplicación, por lo que declinamos esta opción

Realizamos otra investigación y observamos que existía una gran cantidad de IDS configurables, tanto Open Source como SAAS (Software As A Service). Nuestra idea era añadirle a un IDS (Open Source) funcionalidades que dotaran una mayor usabilidad, una configuración más robusta o la creación de reglas a través de machine learning. Esto nos ponía como hándicap que, además de

aprender machine learning, deberíamos comprender el funcionamiento completo de cualquier IDS que eligiéramos.

Tras la búsqueda de varios HIDS, nos inclinamos por SolarWinds Security Event Manager (SEM), que proporciona herramientas automatizadas de corrección de amenazas. Era buena elección, ya que ayudaría a la configuración de la automatización de reglas con machine learning, una de las ideas que estaban planteadas. Este software nos hubiese permitido aprender el funcionamiento y los aspectos en los que se basan para configurar las reglas, para después poder adaptar este proceso a nuestro antojo. Pero su principal problema es que no es una herramienta open-source, por lo que indagar en su programación y modificarla no está permitido.

Los otros candidatos fueron Tripwire, el cual descartamos por su poca documentación, y por no notificar en tiempo real. O Samhain que tenía una amplia documentación pero como en la anterior ocasión, no dispone de análisis a tiempo real y la interfaz para mostrar los datos era muy simple.

Al final nos decantamos por OSSEC por todas las funcionalidades que ofrece y además por ser uno de los HIDS más usados de todos, teniendo una gran reputación en este sector. Las características que conforman a este, como su funcionamiento (que les alza como uno de los HIDS con mejores resultados detectando anomalías) , y por ser de código abierto. Esta herramienta posee de una gran comunidad de contribuidores y una documentación bien detallada. Indagando en OSSEC nos encontramos que contaba con una WUI propia (recomendada en su documentación).

La primera dificultad que encontramos es que es anticuada y desactualizada. La información que muestra no es intuitiva ni descriptiva, así como una navegación incómoda para un usuario estándar. No proporciona las estadísticas de una forma limpia y comprensible. Además de no poder añadir reglas personalizadas desde la web, teniendo que hacerlo mediante comandos, siendo esto una tarea pesada y laboriosa, ya que tienes que saber exactamente qué comando y argumentos utilizar.

Además tuvimos diversos problemas con los permisos de la aplicación, ya que en múltiples ocasiones no funcionaba la web por no ser propietarios de los ficheros, o por no tener autoridad para modificarlos o leerlos.

En definitiva, observamos que OSSEC tenía la necesidad de una interfaz nueva, con más funcionalidades y que fuese más intuitiva. Por esta razón, cambiamos nuestra idea inicial para crear una WUI de mayor calidad, aprovechando los datos que OSSEC nos proporcionaba, manteniendo la estructura de la WUI original pero con un nuevo aspecto, mejor prestaciones y funcionalidades.

2.4 Objetivos finales

Todos estos objetivos se realizarán sobre la WUI de OSSEC:

- Mejorar visualización de amenazas y alertas que OSSEC nos proporciona.
- Añadir la funcionalidad de crear nuevas alertas y monitorización de archivos.
- Mostrar gráficos que ayuden tanto a los administradores de sistemas como a sus superiores, a sacar conclusiones sobre las debilidades de los sistemas y sus puntos más críticos.

2.5 Tareas

Antes de realizar el proyecto, plantemos una serie de tareas a realizar, para llevar un control y una mejor organización del trabajo.

- ❏ **Tarea 1**: Realizar búsquedas de HIDS que se amolden a nuestros requisitos y conocimientos
- ❏ **Tarea 2**: Instalación de HIDS
- ❏ **Tarea 3**: Análisis teórico profundo de la herramienta elegida
- ❏ **Tarea 4**: Análisis del código de la herramienta elegida
- ❏ **Tarea 5**: Creación de nuestra propia documentación de la herramienta
- ❏ **Tarea 6**: Búsqueda de una nueva interfaz
- ❏ **Tarea 7**: Estructuración de nuestro futuro código

3. Descripción de la herramienta OSSEC

En este capítulo describimos en detalle la arquitectura y funcionamiento de la plataforma OSSEC. Sacar el máximo provecho de sus funcionalidades ha representado una parte importante del tiempo que hemos dedicado a este trabajo. Además, el estudio de ingeniería inversa que hemos realizado, nos ha servido de base para luego poder llevar a cabo todas las mejoras implementadas que se describirán en capítulos posteriores.

3.1 ¿Qué es?

OSSEC (*Open Source HIDS SECurity*) [4] es una herramienta de código abierto y de licencia GNU que integra el análisis de registros, monitorización de integridad de archivos, detección de rootkits, respuesta activa de ataques, auditoría de registros de los equipos Windows, monitorización de integridad de archivos, alerta en tiempo real y recopilación de información del sistema. OSSEC está preparado para poder ejecutarse en los siguientes sistemas operativos:

- GNU / Linux (todas las distribuciones, incluidas RHEL, Ubuntu, Slackware, Debian, etc.)
- Windows XP, 2003, Vista, 2008, 2012
- VMWare ESX 3.0,3.5 (incluidas las comprobaciones de CIS)
- FreeBSD (todas las versiones actuales)
- OpenBSD (todas las versiones actuales)
- NetBSD (todas las versiones actuales)
- Solaris 2.7, 2.8, 2.9, 10 y 11.4
- AIX 5.2 y 5.3
- Mac OS X 10.x
- HP-UX 11

Los desarrolladores del HIDS lo definen como: "*A scalable, multi-platform, open source Host-based Intrusion Detection System (HIDS)*".

Este software es adaptable según las necesidades de seguridad, ya que proporciona diversas opciones de configuración, inclusión y personalización de reglas y alertas. Además posibilita las monitorizaciones basadas en agentes y sin agentes (como los cortafuegos). También realiza comprobaciones de integridad de los ficheros.

3.2 Beneficios clave

En este apartado describiremos las principales características de OSSEC, que hacen que este software, sea uno de los HIDS más completos. Explicaremos las alertas, tanto su detección como su configuración, y además hablaremos de su gestión centralizada y de sus dos tipos de supervisión .

Alerta y detección

El sistema OSSEC permite detectar y alertar sobre modificaciones no autorizadas del sistema de archivos y comportamiento malicioso del sistema. También cubre las secciones de monitorización de integridad de archivos, inspección y monitorización de registros, y aplicación/verificación de políticas.

Alertas configurables y en tiempo real

Este HIDS admite la configuración de incidentes sobre los que desean recibir alertas, de esa manera se permite al administrador de sistema centrarse en aumentar la prioridad de los incidentes críticos, además éste dispone de opciones de respuesta activa para bloquear un ataque de inmediato. Permite el envío de alertas a través del correo electrónico gracias a la integración con smtp, sms y syslog.

Gestión centralizada

OSSEC proporciona un servidor de administración centralizado para gestionar políticas en múltiples sistemas operativos. Además, permite a los usuarios especificar ciertas invalidaciones al servidor para algunas políticas.

Supervisión de agente y sin agente

OSSEC ofrece flexibilidad en la monitorización de sistemas y componentes de red, basados en agentes y sin agentes, como routers y cortafuegos. La monitorización sin agentes permite a los clientes que tienen restricciones en el software, satisfacer las necesidades de seguridad y cumplimiento.

3.2 Arquitectura

En esta sección hablaremos de la estructura interna del HIDS y los componentes que la constituyen. OSSEC se compone de un administrador (servidor) central para la monitorización y recepción de información de agentes, syslog, bases de datos, y de dispositivos sin agentes. A continuación, explicaremos los distintos tipos de elementos que establecen la estructura del software:

Administrador (servidor)

El administrador es la pieza central del despliegue de OSSEC. Este almacena las bases de datos de comprobación de integridad de archivos, los registros del sistema, eventos y entradas de auditoría al

sistema. Todas las reglas, decodificadores y opciones de configuración principales se almacenan centralmente en el servidor, facilitando la administración incluso con un gran número de agentes.

Agentes

El agente es un pequeño programa, o colección de programas, instalado en los sistemas que van a ser monitorizados. El agente recogerá información y la remitirá al administrador para analizarla (algunas se recogen en tiempo real). Requiere un uso de memoria y de CPU mínimo por defecto, que no afecta al rendimiento del sistema.

En cuanto a la seguridad, el agente funciona como un usuario de privilegios bajos y dentro de una *chroot jail* aislada del sistema. La mayor parte de la configuración del agente puede ser configurada desde el administrador.

OSSEC sólo puede instalarse como agente en plataformas Microsoft Windows. Estos sistemas requerirán un servidor OSSEC, ejecutándose en Linux u otro sistema unix.

Sin Agente

En el caso de los sistemas en los que no se puede instalar un agente, el soporte sin agente puede permitir que se realicen controles de integridad. Los escaneos sin agente se pueden utilizar para monitorizar cortafuegos, routers e incluso sistemas Unix.

Los tres tipos explicados anteriormente son los principales, pero también existen ciertos casos especiales de estos, como por ejemplo la instalación en máquinas virtuales, cortafuegos, interruptores o routers.

Virtualización/Vmware

También se puede instalar dentro de algunas versiones de Vmware ESX, pero esto puede causar problemas de soporte. Con el agente instalado dentro de Vmware ESX se puede obtener alertas sobre cuándo la máquina virtual se está instalando, eliminando, iniciando, etc . También monitoriza los accesos, logouts y errores dentro del servidor ESX. Además, OSSEC realiza las comprobaciones del Center for Internet Security (CIS) para Vmware, alertando si hay alguna opción de configuración insegura habilitada o cualquier otro problema.

Cortafuegos, interruptores y routers

OSSEC puede recibir y analizar eventos syslog desde una gran variedad de cortafuegos, interruptores y routers. Soporta todos los routers Cisco, Cisco PIX, Cisco FWSM, Cisco ASA, Juniper Routers, Netscreen firewall, Checkpoint y muchos otros.

Figura 3.1 Arquitectura Ossec

En la Figura 3.1 podemos ver la estructura de OSSEC. Este diagrama muestra al administrador central recibiendo eventos de los agentes y registros del sistema desde dispositivos remotos.

3.3 Métodos y funcionalidades

Como destacamos en la descripción de este software, OSSEC tiene ciertas ventajas que nos ayudan a la seguridad informática, como la detección de rootkits, la integridad de los ficheros, personalización de reglas... En este bloque explicaremos los componentes que se ocupan de realizar estas funcionalidades.

3.3.1 Agentes

Hay dos tipos de agentes dentro de OSSEC: Los agentes que son instalables y los sin agente. Los primeros son instalados en los host e informan al servidor central a través de un protocolo cifrado, Los segundos no requieren una instalación en host remotos, sino que son procesos iniciados desde el mismo administrador de OSSEC. Los host encargados de recoger los resultados, recopilan información de los sistemas remotos y utilizan métodos RPC (*Remote Procedure Call*)

- Agentes instalables:

  La comunicación entre los agentes y el servidor OSSEC generalmente ocurre en el puerto 1514/udp en modo seguro. Si usa el modo syslog para *ossec-remote* , entonces el puerto 514 es el predeterminado (se admiten tanto UDP como TCP). Estos puertos son configurables en la sección remota de ossec.conf.

  Ejemplo de configuración:

```
<agent_config  os = "Linux" >
    <localfile> … </localfile>
    <rules> … </rules>
    …
</agent_config>
```

- Sin agentes

  La supervisión sin agente le permite ejecutar la verificación de integridad en sistemas sin un agente instalado. Es muy útil para chequear, por ejemplo, cortafuegos o routers donde no se puede instalar OSSEC.

  Estas son las opciones de configuración:

  - *type*: El tipo de verificación que se ejecutará en el sistema sin agente. Opciones disponibles:
    - ssh_integrity_check_bsd , ssh_integrity_check_linux, ssh_generic_diff, ssh_pixconfig_diff.
  - *frequency*: Controla el número de segundos entre cada ejecución.
  - *host*: Define el nombre de usuario y el host sin agente.
  - *state*: Esto determina las comprobaciones. Existe de dos tipos:.
    - periodic: la salida de los scripts es procesada por los procesos de OSSEC.
    - periodic_diff: la salida de los scripts se compara con la salida de ejecuciones anteriores.
  - *arguments*: Define los argumentos pasados al script.

  Ejemplo de configuración:

```
<agentless>
    <type> ssh_integrity_check_linux </type>
    <frequency> 36000 </frequency>
    <host> root@linux.server.example.com </host>
    <state> periodic </state>
    <arguments> /bin/etc//sbin </argumentos>
</agentless>
```

3.3.2 Análisis/monitorización de registros

OSSEC recopila los eventos gracias a su recolector de registros (logcollector) y los decodifica, filtra y clasifica realizando así un análisis (gracias al proceso analysisd). Puede leer los eventos desde archivos de registros internos, desde eventos de Windows, o recibirlos a través de un syslog remoto. Para configurarlo [5] se debe especificar las opciones en el archivo ossec.conf de cada agente o en el recurso compartido agent.conf.

Ejemplo de configuración:

```
<localfile>
    <log_format>command</log_format>
    <command>uptime</command>
</localfile>
<localfile>
    <location>/var/log/messages</location>
    <log_format>syslog</log_format>
</localfile>
```

En OSSEC existen dos métodos principales para monitorizar registros: archivo y proceso.

Monitorización de procesos: OSSEC trata todo como si fuera un registro y con ello consigue analizar según las reglas estimadas. Sin embargo, parte de la información no está disponible en los archivos de registro. Para poder resolver este problema, se puede agregar la monitorización de la salida de comandos y tratar la salida de esos comandos como si fueran archivos de registro.

Monitorización de archivos: OSSEC tiene el proceso ossec-logcollector que monitoriza los archivos de registro configurados para nuevos eventos. Cuando llegan nuevos mensajes de registro, los reenvía a otros procesos para su análisis o transporte a un servidor OSSEC.

3.3.3 Syscheck

Syscheck es el nombre del proceso de verificación de integridad dentro de OSSEC. Se ejecuta periódicamente para verificar si algún archivo configurado (o entrada de registro en Windows) ha sido modificado mediante la comprobación de su huella digital. La comprobación de integridad es una parte esencial de la detección de intrusos, que detecta cambios en la integridad del sistema. OSSEC lo hace buscando cambios en las sumas de verificación MD5 / SHA1 de los archivos clave del sistema, y también en el registro de Windows. El agente escanea el sistema cada pocas horas (definido por el usuario) y envía todas las sumas de verificación al servidor. El servidor almacena las sumas de comprobación y busca modificaciones en ellas. Se envía una alerta si algo cambia. Ossec-syscheck es capaz de verificar la integridad del archivo en tiempo casi real en Windows y en las distribuciones modernas de Linux.

Opciones de configuración

Las opciones de configuración [6] se pueden especificar en el archivo ossec.conf de cada agente, a excepción de auto_ignore y alert_new_file que se aplican al administrador y a las instalaciones locales.

Ejemplo de configuración:

```
<syscheck>
    <directories check_all="yes">/etc,/usr/bin,/usr/sbin</directories>
```

```
</syscheck>
```

Monitorización en tiempo real

OSSEC admite monitorización de integridad de archivos en tiempo real (de forma continua) en Linux y sistemas Windows. Primero, el demonio ossec-syscheck necesita escanear el sistema de archivos y agregar cada subdirectorio a la cola en tiempo real. Esto puede tardar un tiempo en finalizar. Tanto rootcheck como syscheck se ejecutan en el mismo subproceso. Cuando se ejecuta rootcheck, los eventos de syscheck se ponen en cola hasta que finaliza.

Base de datos de la lista blanca MD5

Ossec-analysisd puede consultar una base de datos sqlite para hashes md5 conocidos. La base de datos debe contener los hashes, nombres de archivo y una fecha opcional.

3.3.4 Rootcheck

OSSEC realiza la detección de rootkits en todos los sistemas donde esté instalado el agente. El rootcheck (motor de detección de rootkit) se ejecutará cada X minutos (especificado por el usuario, por defecto cada 2 horas) para detectar cualquier posible rootkit instalado. Utilizado con el análisis de registro y el motor de verificación de integridad, se convierte en una solución de monitorización muy poderosa. Rootcheck tiene su propia sintaxis de configuración [7]

Comprobaciones que realiza OSSEC:

1. Lee el archivo rootkit_files.txt que contiene una base de datos de rootkits y archivos comúnmente utilizados por ellos. Utiliza todas estas llamadas al sistema porque algunos rootkits a nivel de kernel ocultan archivos de algunas llamadas al sistema.
2. Lee el rootkit_trojans.txt que contiene una base de datos de firmas de archivos troyanos. Este método de detección no encontrará ningún rootkit a nivel de kernel ni ningún rootkit desconocido.
3. Escanea el directorio */dev* buscando anomalías. El directorio */dev* solo debe tener archivos de dispositivo y el script Makedev. Muchos rootkits usan */dev* para ocultar archivo
4. Escanea todo el sistema de archivos en busca de archivos inusuales y problemas de permisos. Los archivos propiedad de root con permiso de escritura para otros son muy peligrosos por lo que el detector de rootkits lo detectará
5. Busca la presencia de procesos ocultos. Usa getsid() y kill() para verificar si se está usando cualquier pid o no
6. Busca la presencia de puertos ocultos. Usa bind() para verificar cada puerto tcp y udp en el sistema
7. Escanea todas las interfaces en el sistema y busque las que tengan habilitado el modo promiscuo. Si la interfaz está en modo promiscuo, la salida de "ifconfig" debería mostrarlo. Si no, probablemente tenga un rootkit instalado.

3.3.5 Reglas y decoders

En OSSEC, los decodificadores intentan analizar un mensaje del registro, extrayendo información importante para su uso en otros lugares. La información (nombres de usuario o las direcciones IP) se puede traspasar a las reglas para compararla, si no coincide con el log final detectaremos una anomalía.

Las reglas están agrupadas según su tipo de gravedad. Existen diferentes tipos de niveles:

- **Nivel 0 (Ignorado - No se toman medidas)**. Se usa para evitar falsos positivos. Estas reglas se escanean antes que todas las demás. Incluyen eventos sin relevancia de seguridad.
- **Nivel 2 (Notificación de baja prioridad del sistema o notificación del sistema o mensajes de estado)**: No tienen relevancia de seguridad.
- **Nivel 3 (Eventos exitosos / autorizados)**: Incluyen intentos exitosos de inicio de sesión, firewall permite eventos, etc.
- **Nivel 4 (Error de baja prioridad del sistema)**: Errores relacionados con configuraciones incorrectas o dispositivos / aplicaciones no utilizados. No tienen relevancia de seguridad y generalmente son causadas por instalaciones predeterminadas o pruebas de software.
- **Nivel 5 (Error generado por el usuario)**: incluyen contraseñas perdidas, acciones denegadas, etc. Por sí mismas no tienen relevancia de seguridad.
- **Nivel 6 (Ataque de baja relevancia)** : indican un gusano o un virus que no tienen ningún efecto en el sistema (como el código rojo para servidores apache, etc.). También incluyen eventos IDS frecuentes y errores frecuentes.
- **Nivel 7 (Detección de palabras "conflictivas")**: Incluyen palabras como "malo", "error", etc. La mayoría de las veces, estos eventos no se clasifican y pueden tener cierta relevancia de seguridad.
- **Nivel 8 (Primera vez visto)**: Incluye eventos vistos por primera vez. La primera vez que se activa un evento IDS o la primera vez que un usuario inicia sesión. Si acaba de comenzar a usar OSSEC HIDS, estos mensajes probablemente serán frecuentes. Después de un tiempo deberían desaparecer, también incluye acciones relevantes para la seguridad (como el inicio de un sniffer o algo así).
- **Nivel 9 (Error de fuente inválida)**: Incluye intentos de iniciar sesión como usuario desconocido o de una fuente inválida. Puede tener relevancia de seguridad (especialmente si se repite). También incluyen errores con respecto a la cuenta "admin" (raíz).
- **Nivel 10 (Múltiples errores generados por el usuario)**: incluyen múltiples contraseñas incorrectas, múltiples inicios de sesión fallidos, etc. Pueden indicar un ataque o simplemente puede ser que un usuario haya olvidado sus credenciales.
- **Nivel 11 (Advertencia de comprobación de integridad)**: incluyen mensajes sobre la modificación de archivos binarios o la presencia de rootkits (por rootcheck). Si acaba de modificar la configuración de su sistema, debería estar bien con respecto a los mensajes "syscheck". Pueden indicar un ataque exitoso. También se incluyen eventos IDS que serán ignorados (gran número de repeticiones).
- **Nivel 12 (Evento de alta importancia: incluyen mensajes de error o advertencia del sistema)**: Pueden indicar un ataque contra una aplicación específica.
- **Nivel 13 (Error inusual)**: La mayoría de las veces coincide con un patrón de ataque común.
- **Nivel 14 (Evento de seguridad de alta importancia)**: La mayoría de las veces se hace con correlación e indica un ataque.

- ❖ **Nivel 15 (Ataque grave)**: No hay posibilidades de falsos positivos. La atención inmediata es necesaria.

Prueba de reglas y decodificadores

El primer problema que se tiene al solucionar problemas o intentar escribir nuevas reglas y decodificadores es cómo probarlos. OSSEC proporciona el proceso ossec-logtest para simplificar esta tarea.

Búsquedas de listas CDB (constant database)

Permite búsquedas de CDB desde dentro de las reglas de configuración en OSSEC (ossec-analysisd). Es útil para cualquier cosa que tenga una gran cantidad de artículos. Algunos ejemplos:

- Nombrado con registros recursivos que comprueban la lista www.malwaredomains.com en busca de dominios sospechosos.
- Listas de usuarios aprobados por servidor.
- Búsqueda de direcciones IP: hay una gran cantidad de listas de direcciones IP erróneas sospechosas o conocidas que coinciden dentro de las reglas ossec.

Sintaxis para listas CDB:

- Reglas: Una regla usaría la siguiente sintaxis para buscar una clave dentro de una base de datos CDB:

```
<list field = "" otros atributos> reglas/registros </list>
```

- ossec.conf: Cada lista necesitará ser definida y se le pedirá que esté disponible usando la siguiente sintaxis:

```
<ossec_config>
  <rules> <list> reglas / registros </list> </rules>
</ossec_config>
```

- Comandos: Los archivos CDB deben compilarse antes de que puedan usarse. El comando ossec-makelists procesará y compilará todas las listas si se han cambiado las reglas del texto maestro.

Crear decodificador personalizado y reglas

1. Agregar un archivo para ser monitorizado en ossec.conf.
2. Crear un decodificador o regla personalizada:
    a. Decoder personalizado: Debemos introducir el decoder que deseamos personalizar dentro del archivo ossec.conf. Estos se configuran a través de diversos atributos [8]. Un ejemplo de decoder personalizado

```
<decoder name="example">
  <order>id</order>
```

```
</decoder>
```

b. Regla personalizada: Debemos introducir la regla que deseamos personalizar dentro del archivo ossec.conf. Las reglas también disponen de sus propios atributos y su propia configuración [9]. Un ejemplo de regla personalizada:

```
<rule id="100000" level="7">
  <list lookup="match_key" field="srcip">path/to/list/file</list>
  <description>Checking srcip against cdb list file</description>
</rule>
```

Ossec-logtest se usará para probar el decodificador personalizado y cualquier regla personalizada. Se agregan decodificadores personalizados al archivo local_decoder.xml, que normalmente se encuentran en /var/ossec/etc. Ossec-anaylistd permite cargar directorios completos de archivos como decodificadores, listas o reglas.

3.3.6 Respuesta Activa

La respuesta activa permite que se tome medidas inmediatas cuando se activan las alertas específicas. Esto puede evitar que se propague un incidente antes de que un administrador pueda tomar medidas. La función de respuesta activa dentro de OSSEC puede ejecutar aplicaciones en un agente o servidor en respuesta a ciertos desencadenantes. Estos desencadenantes pueden ser alertas específicas, niveles de alerta o grupos de reglas.

Crear respuestas activas personalizadas:

La configuración de respuesta activa se divide en dos partes. En el primero, se configura los comandos que desea ejecutar. En el segundo, vincula los comandos a reglas o eventos.

1. **Configuración de comandos**: Puedes tener tantos comandos como desee. Cada uno debe estar dentro de su propio elemento de "command". La configuración de comandos se realiza a través del fichero ossec.conf [10].

   Ejemplo de configuración:

```
<command>
    <name> El nombre (A-Za-Z0-9) </name>
    <executable> El comando a ejecutar (A-Za-z0-9.-) </executable>
    <expect> Lista separada por comas de argumentos (A-Za-z0-9) </expect>
    <timeout_allowed> sí / no </timeout_allowed>
</command>
```

2. **Configuración de respuestas:** En la configuración de respuesta activa [11], vincula los comandos (creados) a los eventos. Puedes tener tantas respuestas como quieras. Cada uno debe estar dentro de su propio elemento de "respuesta activa".

   Ejemplo de configuración:

```
<active-response>

  <command> firewall-block </command>

  <location> agente definido </location>

  <agent_id> 001 </agent_id>

  <rules_group> autenticación_failed, autenticación_failures </rules_group>

  <timeout> 600 </ timeout >

  <repeated_offenders> 30,60,120 </repeated_offenders>

</active-response>
```

3.3.7. Opciones de salida y alerta

OSSEC incluye varias formas de enviar alertas a otros sistemas o aplicaciones. Syslog, correo electrónico y envío de alertas a una base de datos SQL son los métodos típicos. Estos métodos de salida solo envían alertas, no datos de registro completos. Como los agentes no generan alertas, estas opciones son solo del lado del servidor.

Envío de alertas a través de syslog : La salida de Syslog permite que un administrador de OSSEC envíe las alertas a uno o más servidores de Syslog. Debido a que solo envía las alertas a través de syslog, estas opciones son solo para servidores o instalaciones locales.

Envío de alertas por correo electrónico: Actualmente hay tres tipos de alertas por correo electrónico: direcciones de correo electrónico de notificación única, notificaciones granulares a cualquier número de direcciones de correo electrónico e informes diarios por correo electrónico

Almacenar alertas como JSON: Esto puede proporcionar el método más simple de exportar el mensaje de alerta completo a otros programas sin ninguna limitación o dependencia

Envío de salida a una base de datos: OSSEC soporta salidas de bases de datos MySQL y PostgreSQL. Estas opciones de configuración se pueden especificar en el servidor o en el archivo ossec.conf de instalación local [12].

Envío de salida al Prelude: Prelude es un IDS híbrido que utiliza IDMEF para recibir información de alertas de dispositivos externos. OSSEC permite a un usuario enviar sus alertas de OSSEC a Prelude. Podemos observar los pasos en la documentación de OSSEC [13]

3.4 Instalación

En este bloque explicaremos los pasos a seguir para instalar OSSEC de manera correcta, elegir la configuración más adecuada para nosotros y además, explicaremos los requisitos que debemos tener para poder hacerlo.

3.4.1 Requisitos para la instalación

Los siguientes paquetes son necesarios para la instalación de OSSEC:

1. **PCRE2**

Es una API revisada para la biblioteca PCRE, que es un conjunto de funciones, escritas en C, que permite obtener coincidencias mediante patrones de expresión regular.

2. **ZLIB**

Es una biblioteca de software utilizada para compresión de datos. Es una abstracción del algoritmo DEFLATE de compresión utilizado en el programa de compresión del archivo gzip . zlib

3. **LIBEVENT**

Esta API permite ejecutar "callbacks" cuando un evento ocurre en un descriptor de fichero. Además libevent también nos proporciona esta funcionalidad para señales y timeouts.

3.4.1 Pasos a seguir

La instalación la hemos realizado en una máquina virtual Linux, en concreto la distribución Ubuntu, por lo que, los comandos de instalación que explicaremos a continuación serán sobre este sistema operativo.

La versión de OSSEC que hemos instalado es la 3.6.0, es la versión estable de el software más reciente, la podemos obtener desde el siguiente enlace :

```
wget https://github.com/ossec/ossec-hids/archive/3.6.0.tar.gz
```

El archivo comprimido con los programas necesarios para la instalación de OSSEC viene con un script de instalación que nos guía durante todo el proceso mediante preguntas sobre preferencias de instalación. Para elegir una instalación correctamente, primero debemos de entender los distintos tipos de esta que existen. OSSEC se puede instalar en una combinación de agente/servidor o como un sistema independiente. La instalación independiente es esencialmente una instalación de servidor sin las piezas que interactúan con los agentes. La instalación del servidor incluye la funcionalidad del agente para el sistema local. Además de los tipos servidor y agente, el programa nos permite elegir otros dos tipos de instalación algo más peculiares.

Híbrida
Una instalación híbrida es tanto un servidor como un agente. Como servidor procesa registros para un número de agentes, y como agente envía alertas a otro servidor.

Local
Una instalación local, o independiente, reside íntegramente en un sistema singular. No está asociada con un servidor o agentes. Los decodificadores y las reglas se almacenarán en una instalación local.

En nuestro caso hemos hecho realizado la instalación de tipo local. Hemos elegido este tipo para poder tener toda la información que genera OSSEC en una sola máquina, y así poder acceder a esta de una manera más sencilla si lo necesitamos.

En la instalación también bién nos deja configurar otros factores como:

- Recibir alertas por correo electrónico.
- Agregar sistema de detección de rootkit
- Activar la respuesta activa.
- Agregar una IP a la lista blanca

3.4 Componentes

Como parte de nuestro trabajo hemos hecho un estudio de los componentes para comprender el funcionamiento interno de OSSEC. Para ello vamos a hacer una descripción de los ficheros que encontramos en esta herramienta y los daemons que utiliza para ejecutar cada funcionalidad. Los directorios y archivos principales que componen OSSEC son los siguientes:

- **/var/ossec/bin**: Binarios usados por OSSEC. Estos binarios sirven para ejecutar cada componente de OSSEC y realizar todas las funcionalidades que ofrece este programa.
- **/var/ossec/etc**: Directorio donde se almacenan los archivos de configuración.
- **ossec.conf**: El archivo principal de configuración. Aquí se almacenan todos los ficheros que deseamos monitorizar y con qué tipo de monitorización vamos a llevar a cabo con estos. Más adelante se darán detalles de como configurarlo.
- **internal_options.conf**: Configuraciones adicionales.
- **client.keys**: Claves de autenticación entre cliente y servidor.
- **/var/ossec/logs**: Directorio con los logs relacionados con OSSEC.
- **ossec.log**: Archivo principal de log de OSSEC.
- **alerts/alerts.log**: Logs de alerta de OSSEC.
- **active-responses.log**: Logs del módulo de respuesta activa de OSSEC.
- **/var/ossec/queue**: Directorio con los archivos de colas de operaciones de OSSEC.
- **agent-info**: Información específica de los agentes.
- **syscheck**: Chequeos de integridad de los archivos monitorizado por OSSEC.

- **rootcheck**: Información de políticas y rootkits de OSSEC.
- **rids**: ID's de mensajes de los agentes.
- **/var/ossec/stats**: Directorio con archivos de información estadística de OSSEC.
- **var/ossec/rules**: Directorio que contiene las reglas usadas por OSSEC

OSSEC son varios daemons que controlan distintos aspectos, y todos son configurables desde ossec.conf. Los daemons que se ejecutan en OSSEC son los siguientes:

- **analysisd**: Proceso principal, hace todo el análisis. Analiza datos de los registros, syscheck, rootcheck, etc. Recibe los mensajes de registro y los compara con las reglas. Creará alertas cuando un mensaje de registro coincida con una regla aplicable.
- **remoted**: Recibe logs remotos desde los agentes. Puede escuchar el puerto 1514 / udp (para comunicaciones OSSEC) y/o 514 (para syslog). Está configurado en la sección <remote> de ossec.conf.
- **logcollector**: Supervisa los archivos y comandos configurados (syslog, win event log, etc).
- **agentd**: Es el proceso del lado del cliente que se comunica con el servidor.
- **maild**: Se encarga de enviar mails con las alertas.
- **execd**: Ejecuta las respuestas activas con los scripts configurados.
- **monitord**: Monitoriza el estado del agente y comprime los archivos de registro diarios
- **dbd**: Se encarga de insertar los registros de alertas en una base de datos, ya sea postgresql o mysql

4. OSSEC WUI - Front

En este capítulo, explicaremos las distintas vistas de la interfaz de OSSEC, y su evolución a lo largo del desarrollo de nuestro trabajo, haciendo énfasis en las carencias que hemos encontrado en la interfaz oficial, y proponiendo nuevas funcionalidades de esta.

La aplicación original de OSSEC presenta una interfaz básica para poder leer la información de la monitorización del sistema. La WUI consta de los siguientes componentes principales:

- Vista principal (Main).
- Vista de búsqueda de alertas.
- Vista para ver los archivos modificados
- Estadísticas de alertas

Lo primero que vemos cuando abrimos la aplicación original de OSSEC es la vista *Main*:

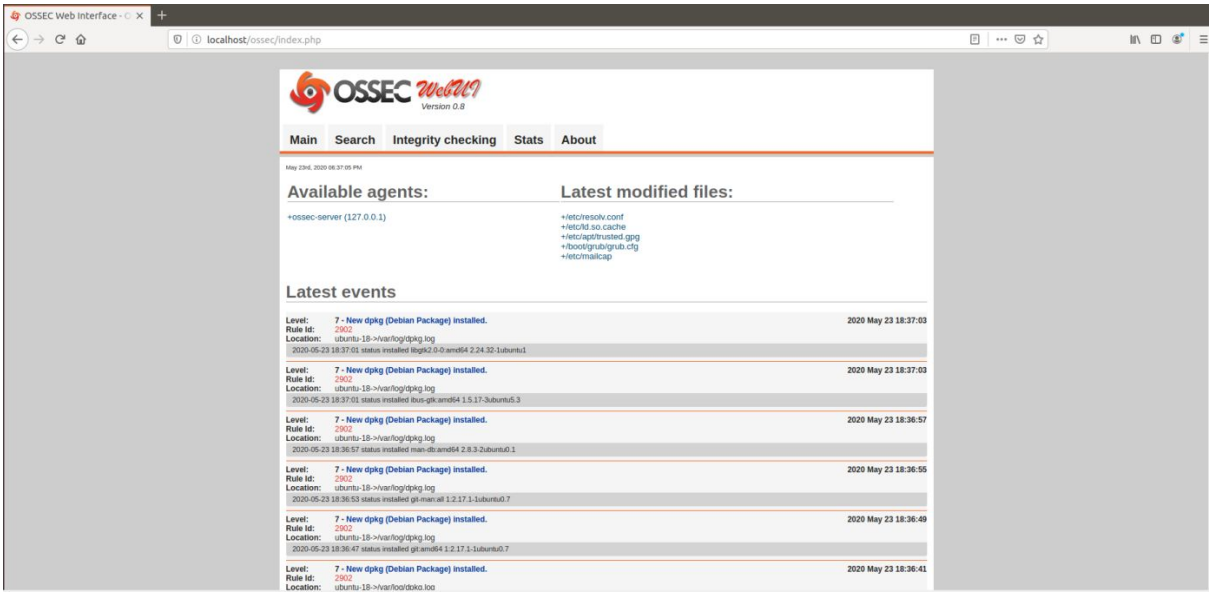

Figura 4.1 WUI original de OSSEC

Como podemos apreciar en la Figura 4.1, es una interfaz que no ha sido adaptada desde que se creó la herramienta OSSEC. Por ejemplo, presenta colores apagados, es poco intuitiva, los laterales son grises... En 2008 la idea de un diseño que se adapte a cualquier resolución, y dispositivo, fue introducida por el W3C, con el fin de poder ofrecer el mismo servicio, con la misma calidad, desde cualquier sitio. Desde ese momento, surgen muchas librerías y diseños que tienen como propósito unificar los procesos de desarrollo de interfaces gráficas, no sólamente para facilitar el trabajo a los desarrolladores (que, en muchos casos, suponía lo contrario), sino para facilitar su uso a los usuarios.

Los propios desarrolladores de OSSEC están de acuerdo en que su interfaz debe ser actualizada, de hecho, en uno de sus archivos incluídos en la instalación de su WUI, aconsejaba que se usaran otras aplicaciones como Kibana, Splunk, u otro proyecto similar para monitorizar alertas, las cuales tienen una interfaz gráfica moderna y actualizada. Sin embargo, la principal desventaja de los proyectos mencionados es que, aunque parte de sus servicios se distribuyen de manera gratuita, solo es de forma temporal, o la documentación ofrecida tiene un acceso muy restringido.

Crear desde cero una WUI no sólo nos ayuda a comprender cómo funciona por completo OSSEC, sino que también nos permite una flexibilidad absoluta a la hora de añadir funcionalidades nuevas.

4.1 Análisis del Front

Antes de modificar cualquier funcionalidad de OSSEC, tenemos que migrar por completo todas las funcionalidades con las que ya cuenta a una nueva interfaz gráfica. Para ello, primero haremos un análisis del *frontend* de la WUI original de OSSEC.

4.1.1 Main

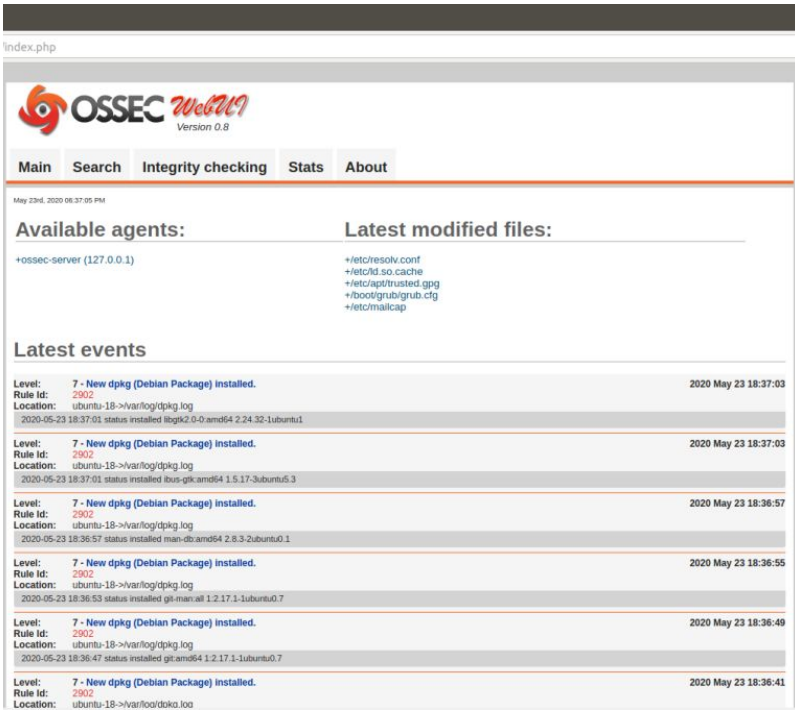

Figura 4.1.1. Página Main de la WUI original de OSSEC

Un problema esencial con el que nos encontramos es que, ni siquiera por la URL, podemos saber en qué pestaña nos encontramos. Cuando cambiamos de pestaña, sobre la página principal *index.php*, se le añaden argumentos para volcar sobre la misma página el módulo pedido.

En esta vista (Figura 4.1.1), podemos encontrar una sección con el resumen de los agentes disponibles (que, por defecto, es el servidor instalado en la propia máquina), y los archivos modificados más recientemente. Al hacer click en ellos, se despliegan datos sobre el fichero, el agente, y la fecha de modificación. En la sección *Latest Events*, muestra una lista de las últimas alertas que se han detectado, así como el nivel de la alerta (cuanto mayor, mayor la gravedad), el ID de la regla, la localización del archivo y la fecha.

4.1.2 Search

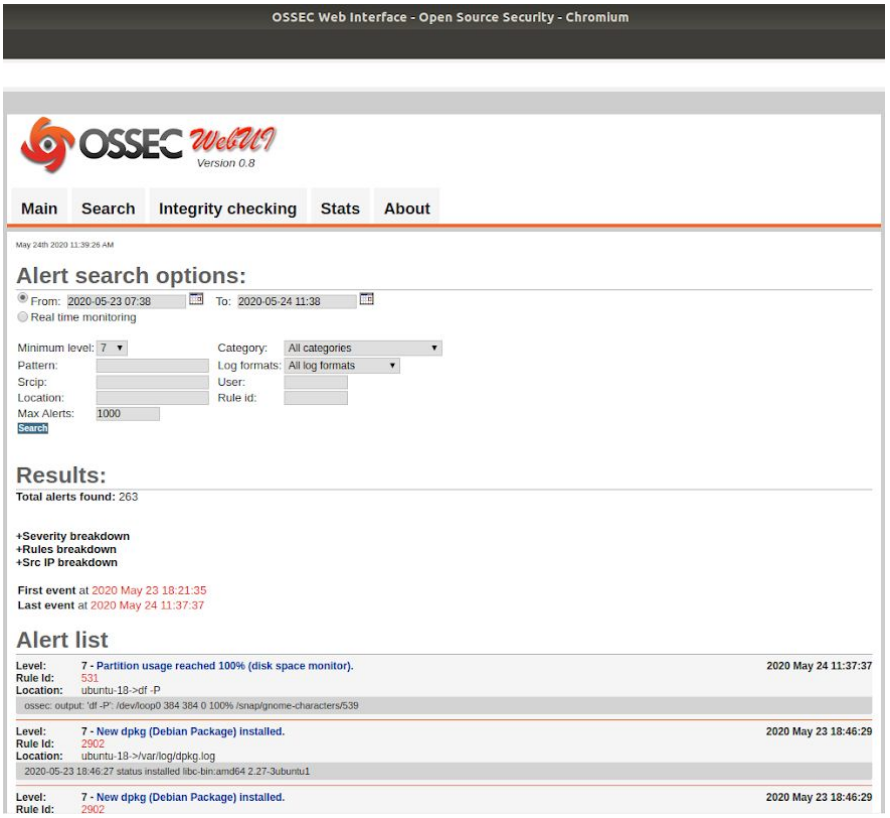

Figura 4.1.2 Página Search de la WUI original de OSSEC

La pestaña de search (Figura 4.1.2) permite filtrar la lista de alertas por nivel, categoría, patrón de caracteres en el mensaje, patrón de strings en el evento, id de la regla, usuario y directorio. También se puede limitar el número de alertas que saldrán en la vista, lo cual, aunque es poco práctico, sirve para analizar con más atención un número definido de alertas.

Por cada alerta que se muestra en pantalla, se puede ver el nivel de la alerta, la causa, el identificador de la regla, la localización del archivo y el último comando que ha recibido el equipo antes de que se haya detectado la anomalía.

4.1.3 Integrity Checking

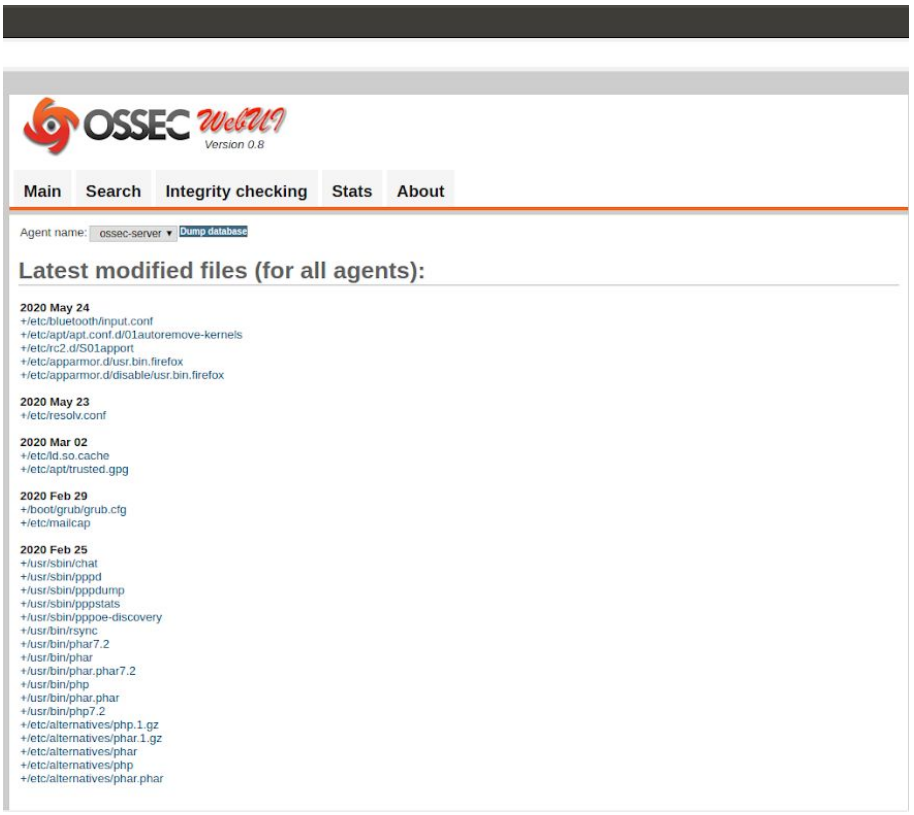

Figura 4.1.3 Página Integrity checking de la WUI original de OSSEC

En esta vista, se muestra un historial con todos los archivos que han sido modificados, organizados por día y mes. Al analizar el código fuente (del cual se hablará con más detalle en el apartado correspondiente) podemos observar que el número de modificaciones que se muestran se limita a 100, lo cual es un exceso innecesario de información.

Al elegir cualquier entrada (Figura 4.1.4), la vista muestra información acerca de la hora, el agente al que pertenecen los archivos de monitorización y el directorio donde se encuentra el fichero.
Si hacemos click en “Dump Database” (Figura 4.1.5), se recarga la misma vista, pero con los datos en otro formato:

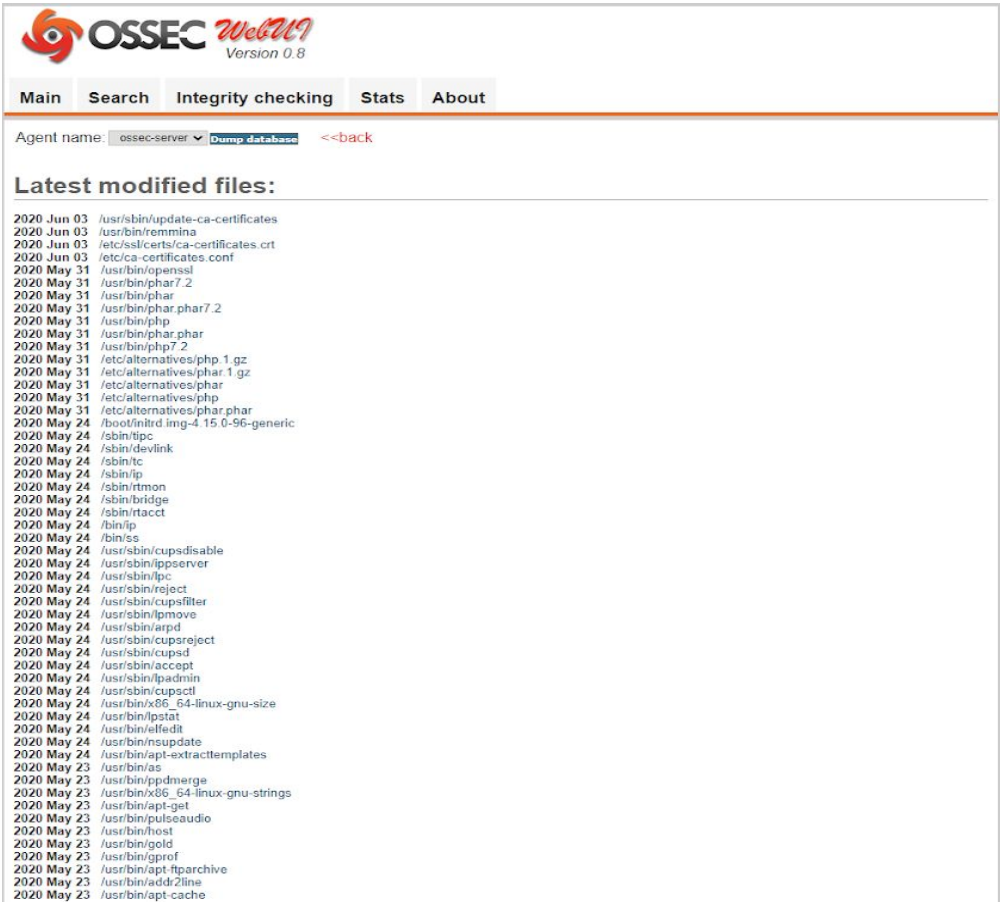

Figura 4.1.4 Listado de archivos, después de hacer dump database

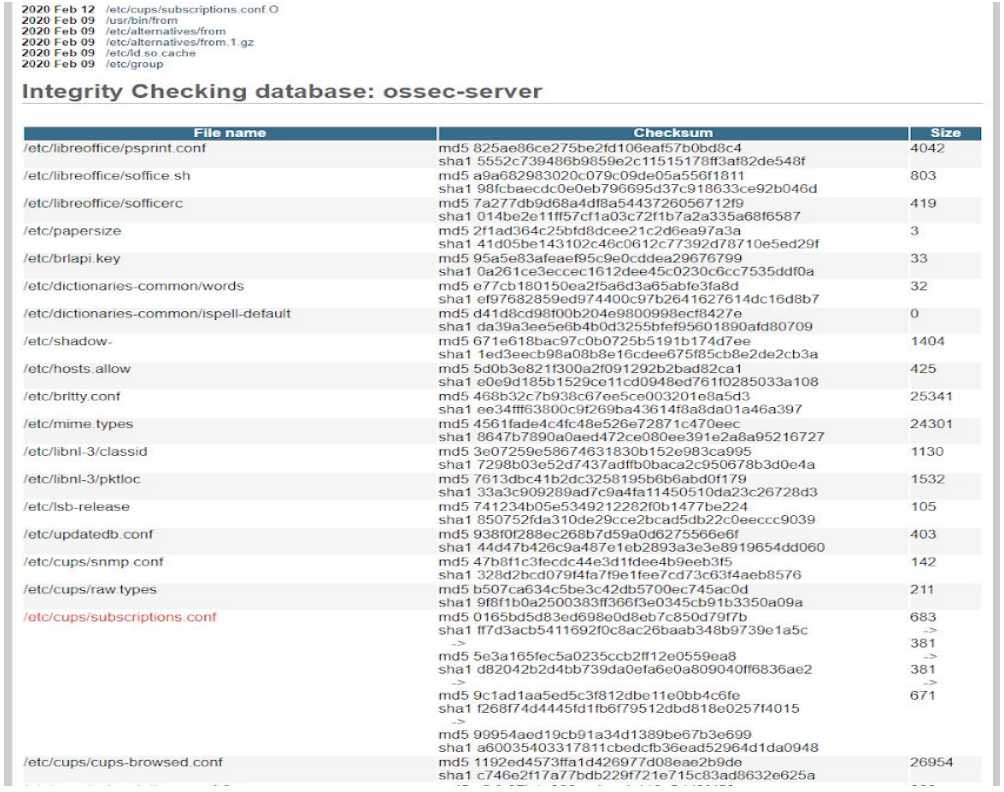

Figura 4.1.5 Historial de hashes para los archivos de la Figura 4.1.4

En la longitud de la lista de archivos de la Figura 4.1.5, podemos ver que hay un exceso de datos. Se imprimen por pantalla absolutamente todas las entradas que existen desde la instalación de OSSEC en el equipo, haciendo que sea muy complicado el análisis de los datos si queremos fijarnos en alguna fecha en concreto.

4.1.4 Stats

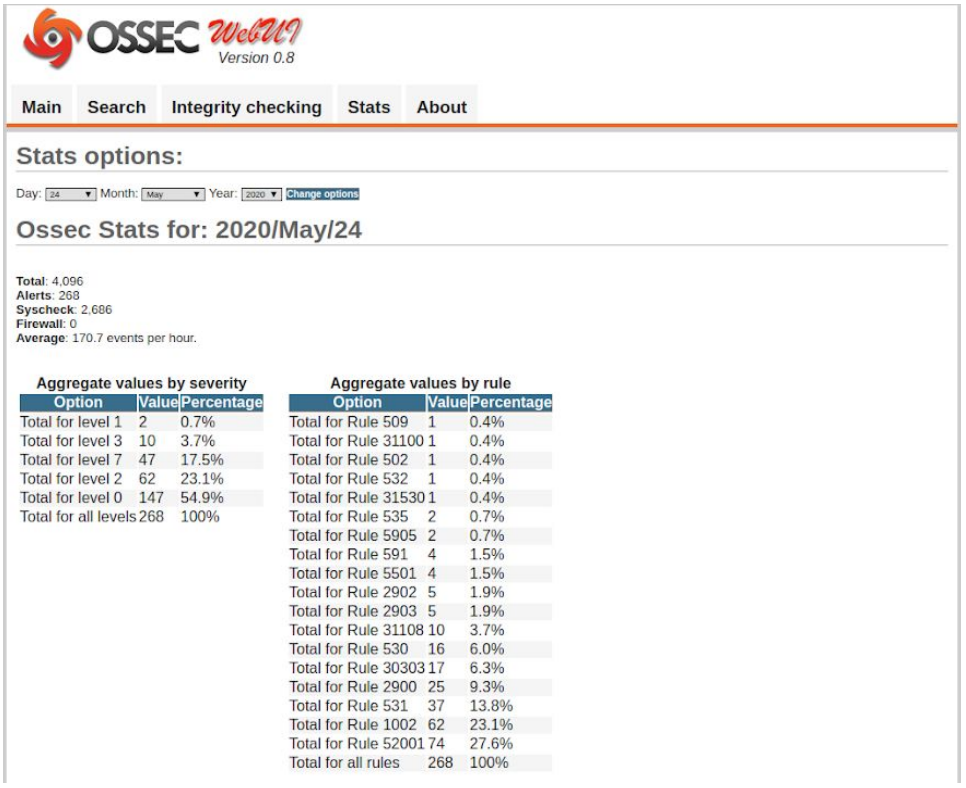

Figura 4.1.6 Página Sats de la WUI original de OSSEC

La vista de la figura 4.1.6 muestra las estadísticas (que pueden ser diarias o del todo mes), de las alertas agrupadas por nivel (gravedad), por regla y por hora. Además de las alertas, también se muestran estadísticas de *syscheks* (procesos de verificación de integridad) y de alertas de *firewall*. El principal problema que se observa es que las tablas están alineadas a la izquierda y usan el estilo por defecto de tabla. Visto de esta forma, no tiene aspecto de estadísticas, sino más bien de un desglose poco detallado de la información recolectada, más parecido a un fichero log en estado puro.

4.1.5 About

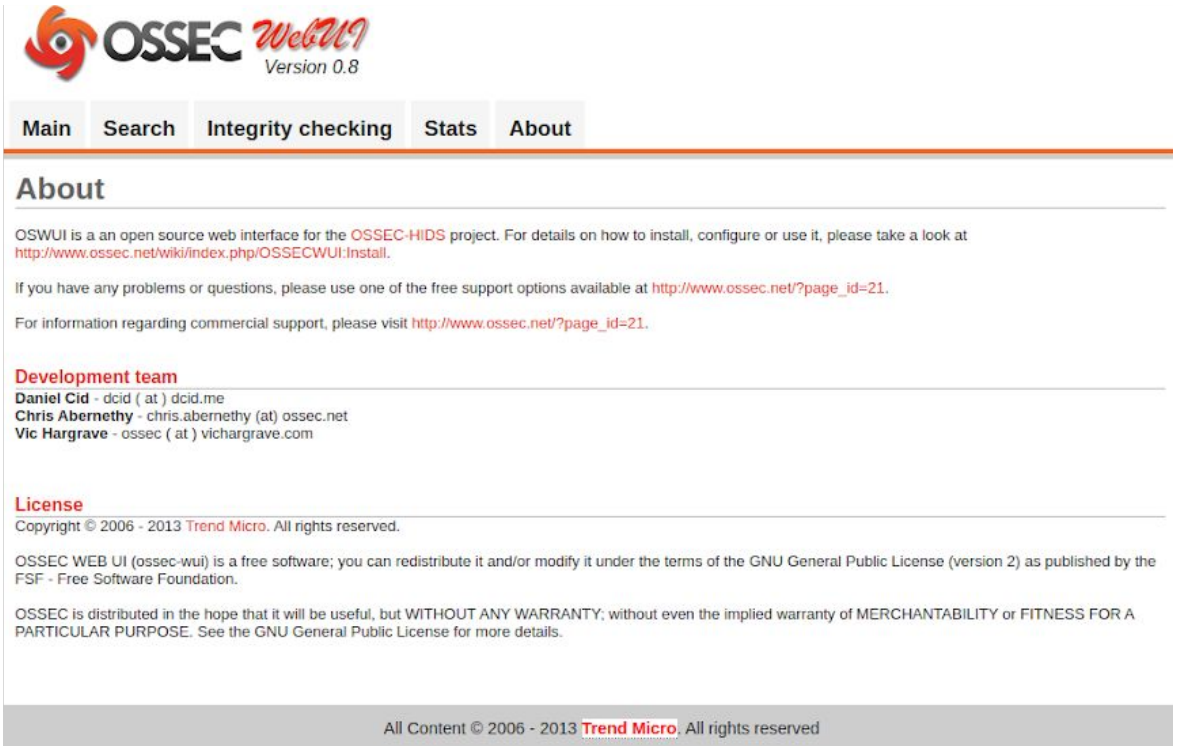

Figura 4.1.7 Página About de la WUI original de OSSEC

Recoge información sobre OSWUI (la WUI que estamos analizando), así como la licencia y los desarrolladores (Figura 4.1.7).

4.2 Mejoras del Front

A día de hoy, cuando navegamos por cualquier página o aplicación web, nos encontramos con una estructura parecida. Esto es debido a que, poco a poco, se ha ido estandarizando el diseño web para incrementar la intuitividad y usabilidad, consiguiendo que cualquier usuario pueda interactuar con los elementos gráficos sin necesidad de un paso previo de aprendizaje. Bien sea mediante el desarrollo de una aplicación web con Wordpress, o programando desde cero, podemos encontrar multitud de librerías que nos ayudan a conseguir este efecto.

Existe una categoría de librerías de frontend de tipo “dashboard” que suelen ser utilizadas para aplicaciones de gestión, administración, o de finanzas, que cumplen con un diseño estándar formado por una barra lateral de acceso rápido, una barra superior con información del usuario y opciones, y una clara división entre los distintos elementos de la página. Debido a la naturaleza de la aplicación que estamos modificando, consideramos que este formato era el más indicado, y por ello, hemos elegido la biblioteca [14] que vamos a describir a continuación.

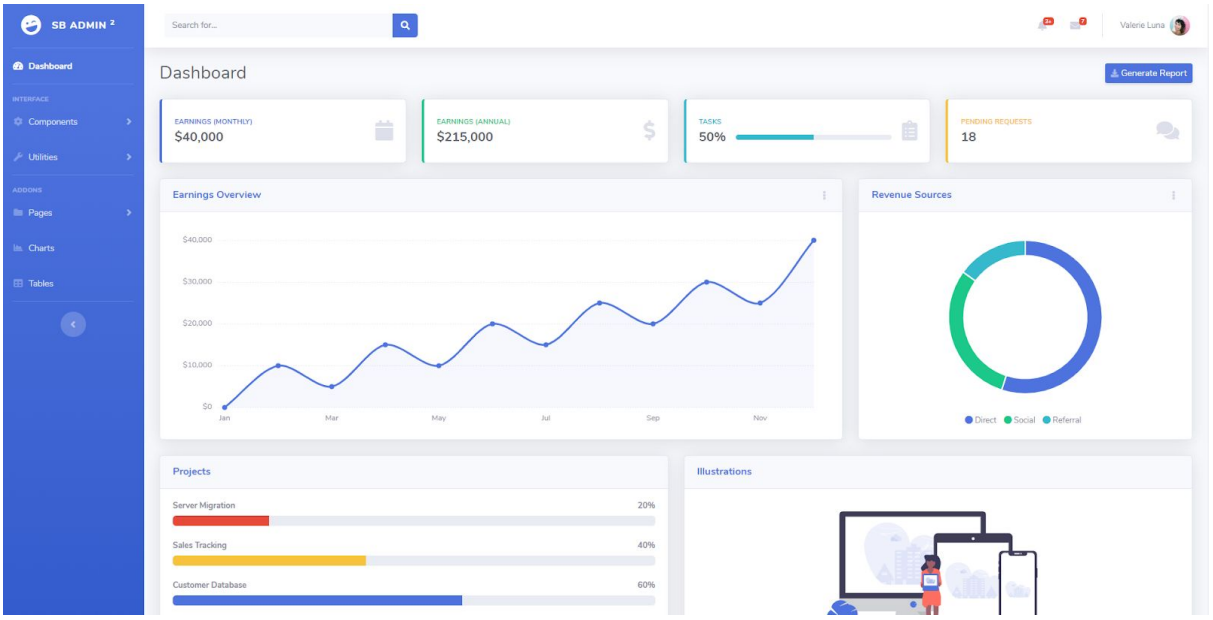

Figura 4.2.1 Plantilla de Bootstrap

Como podemos ver en la Figura 4.2.1, es una interfaz limpia, moderna, y visualmente agradable. No es demasiado compleja, y tiene elementos fáciles de utilizar. Nosotros hemos decidido utilizar la barra lateral y los elementos de "card" a lo largo de la página, a la vez que la librería de gráficas de Google Charts [15] y de tablas de DataTable [16].

En un principio, planteamos la idea de poder iniciar sesión a la aplicación, y dar un uso a la barra superior, otorgando al usuario administrador diversas opciones de gestión de otros usuarios. Pero debido a que nos hemos centrado más en implementar las funcionalidades ya existentes, hemos decidido prescindir de ella.

A continuación, vamos a explicar de forma detallada, vista por vista, los cambios que se han implementado en la nueva WUI. Analizaremos de forma más técnica el desarrollo de la capa frontend en el apartado 4.3.

4.2.1 Home

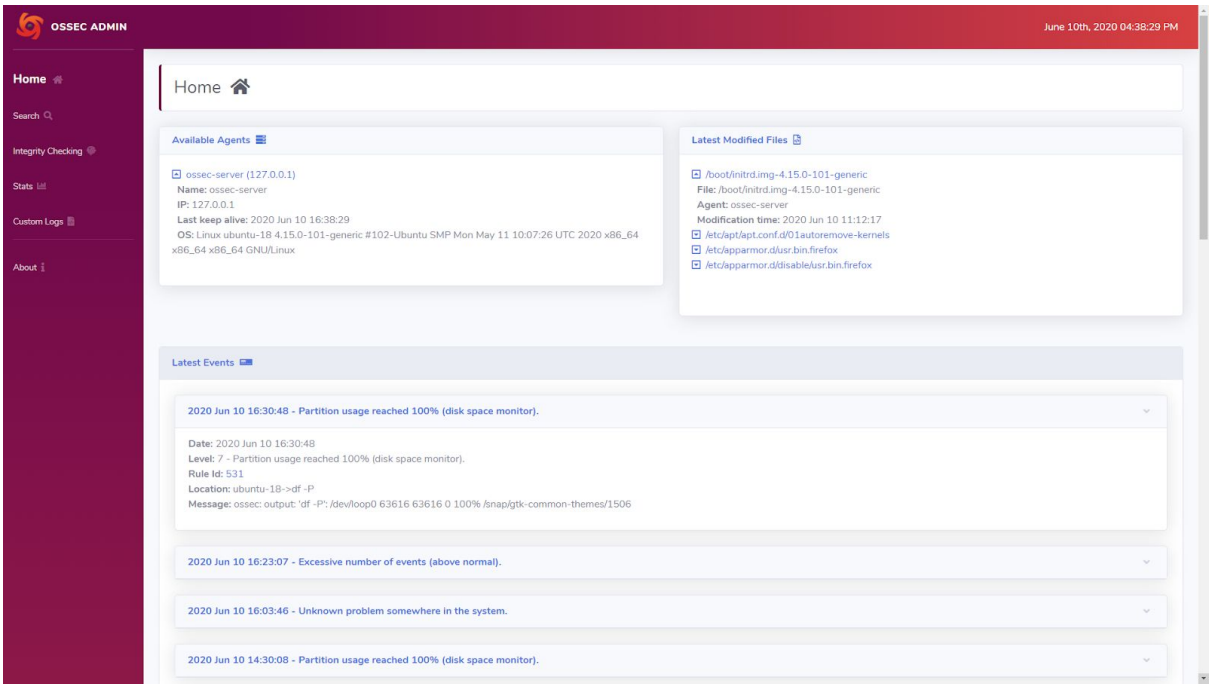

Figura 4.2.2 Página Home de la WUI implementada

Nuestra página de Home, como podemos observar en la Figura 4.2.2, está formada por los mismos elementos que tenía la WUI original, pero es visualmente más comprensible y llamativa. Por un lado, tenemos un barra lateral (visible desde cualquier pestaña de la aplicación), la cual nos indicará en qué página nos encontramos, además de permitirnos acceder de manera rápida a las demás secciones de la aplicación.

El cuerpo de esta pestaña presentan las 3 secciones que aparecían en la versión original. En el apartado de *Last Events* hemos introducido una nueva interacción, como es la expansión de cada bloque de alerta para ver la información. Además, como título aparece la fecha y el nombre del nivel de alerta para hacerla más intuitiva y rápida de buscar. La información que aparece en esta vista es principalmente la misma que encontrábamos en la WUI original distribuida de una forma más atómica y manejable.

4.2.2 Search

Uno de los aspectos más llamativos de una aplicación web es un formulario con un formato moderno y bien estructurado. El problema que presenta un formulario técnico es que muchas veces no sabemos qué estamos escribiendo en él. Por ello, en nuestra implementación, se ha añadido un icono de ayuda al lado de cada entrada que proporciona información con respecto a la sección concreta cuando pasamos sobre ella.

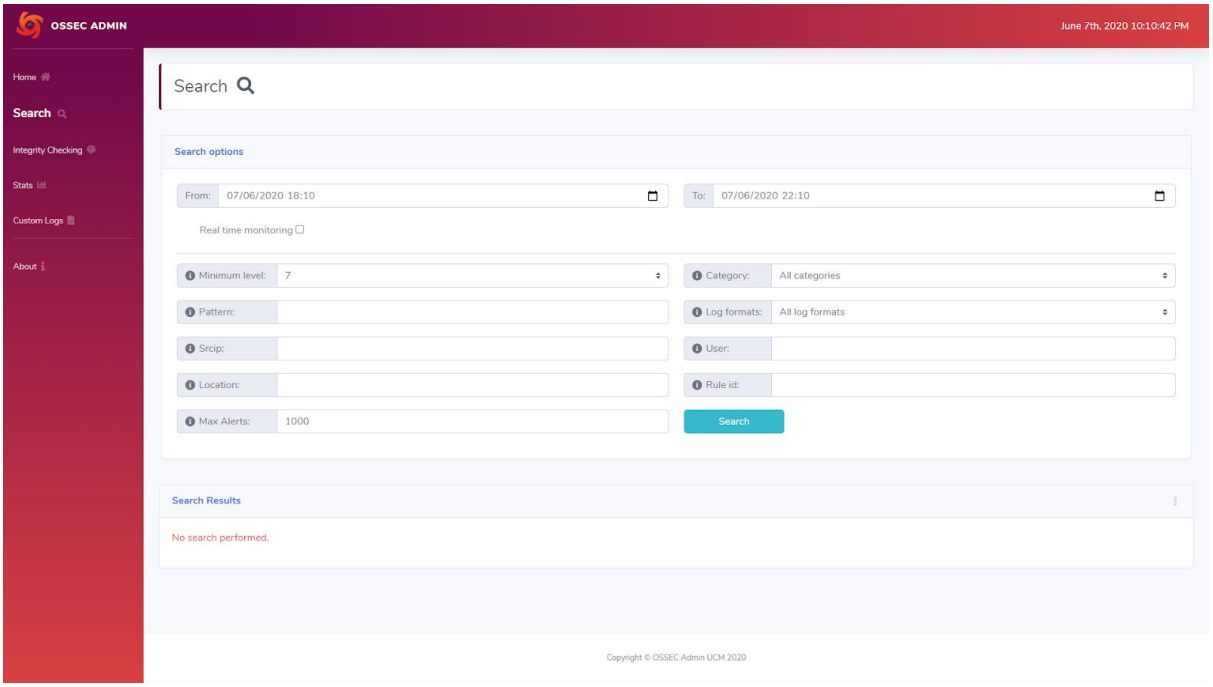

Figura 4.2.3 Página Search de la WUI implementada

En este punto del desarrollo del trabajo no hay ninguna mejora en cuanto a funcionalidad, sólamente en lo visual. Hemos usado una librería actualizada de JavaScript para elegir la fecha de inicio y fin de la monitorización, además de darle un forma más organizada al formulario.

Cuando se hayan rellenado los campos para un búsqueda (mostrados en la figura 4.2.3), y se pulsa el le demos al botón “Search”, aparecerá la información de la siguiente forma (Figura 4.2.4)

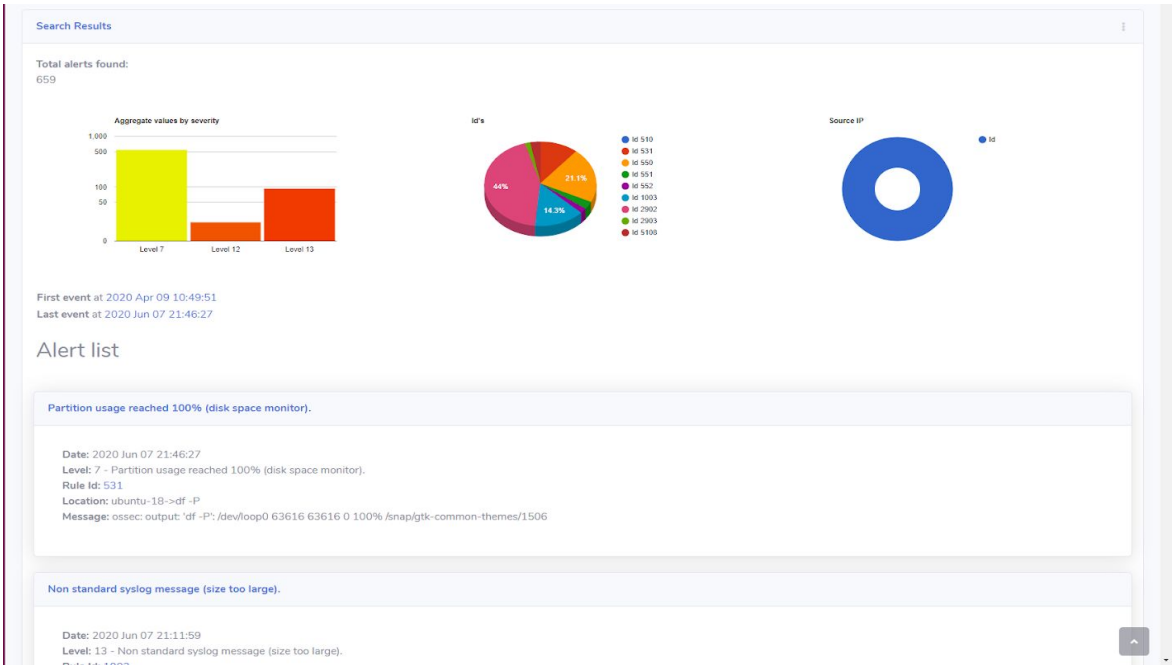

Figura 4.2.4 Datos que muestra la vista Search

Primero de todo, nos encontramos ante una serie de gráficas a modo de resumen, usando la API de Google Charts (de la cual hablaremos con más detalle en el punto 4.2.4). Éstas nos muestran el número de alertas, empezando por la que hemos elegido en la búsqueda (en este caso la alerta mínima es de nivel 7), los ID de las reglas que se han identificado y la IP del origen (que en este caso, es único, y corresponde con el host).

A continuación, tenemos dos líneas que nos indican cuál fue el primer evento desde la fecha elegida hasta el último recogido por la búsqueda (desde el 9 de abril de 2020 hasta el 7 de junio de 2020).

La lista de alertas que nos ha devuelto la petición ahora se representa de forma homóloga al desglose que hacemos en la pestaña *Home*, y será el formato que usaremos por excelencia a lo largo de toda la aplicación. El dividir las alertas por *cards* desplegables nos permite ir cerrando o abriendo aquellas alertas que más nos convengan, y así no tener que entrar en detalle en cada una de ellas.

4.2.3 Integrity Checking

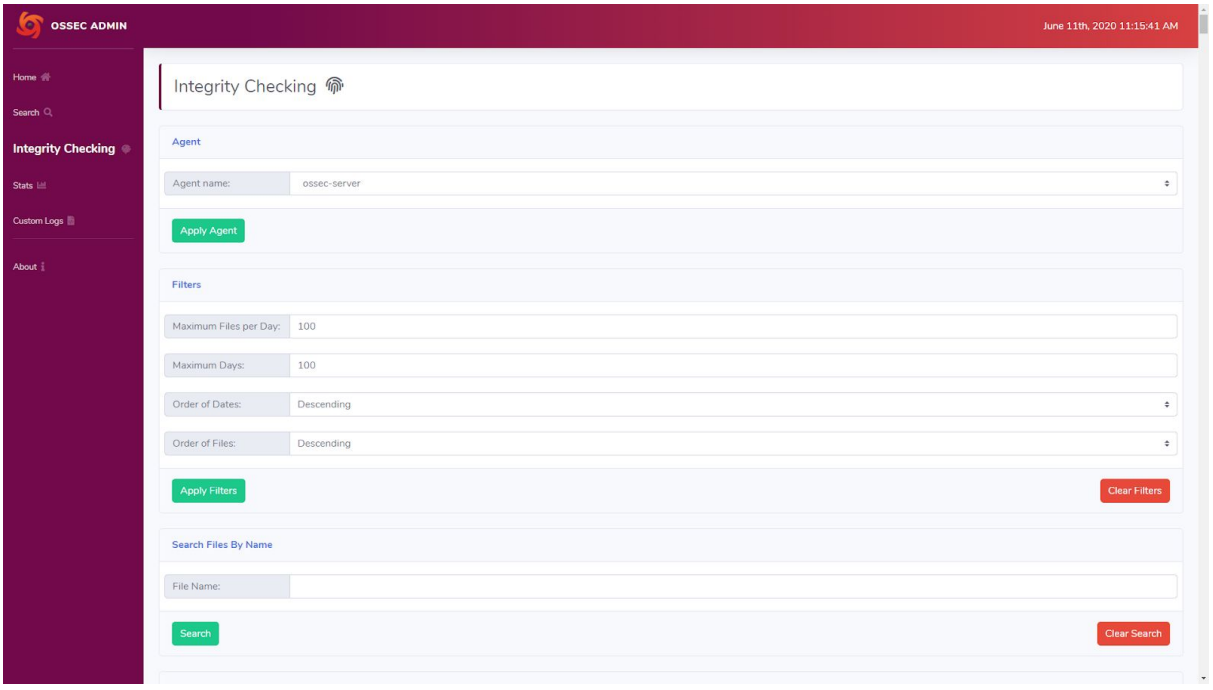

Figura 4.2.5 Página Integrity Checking de la WUI implementada

Anteriormente, esta era una de las vistas menos cuidadas a nivel visual que había, por el exceso de datos que presentaba al mismo tiempo. En esta nueva WUI hemos trabajado para añadir algo muy esencial: un formulario de filtros (Figura 4.2.5). De esa forma, si tan solo queremos ver los datos de hoy, o de esta semana, etc. valdría con rellenar cualquiera de los tres formularios que incluye la vista.

En la figura 4.2.5 se muestra el primer formulario, este consta de las opciones típicas de filtrado. Al haber elegido un día, nos muestra solamente los datos del último día. Si quisiéramos que nos muestre los datos del primer día desde que se instaló OSSEC en el sistema, podemos poner el orden de los archivos como "Ascending", estando por defecto en "Descending". Además, nos permite limitar el número máximo de días y de archivos por día que muestra la vista.

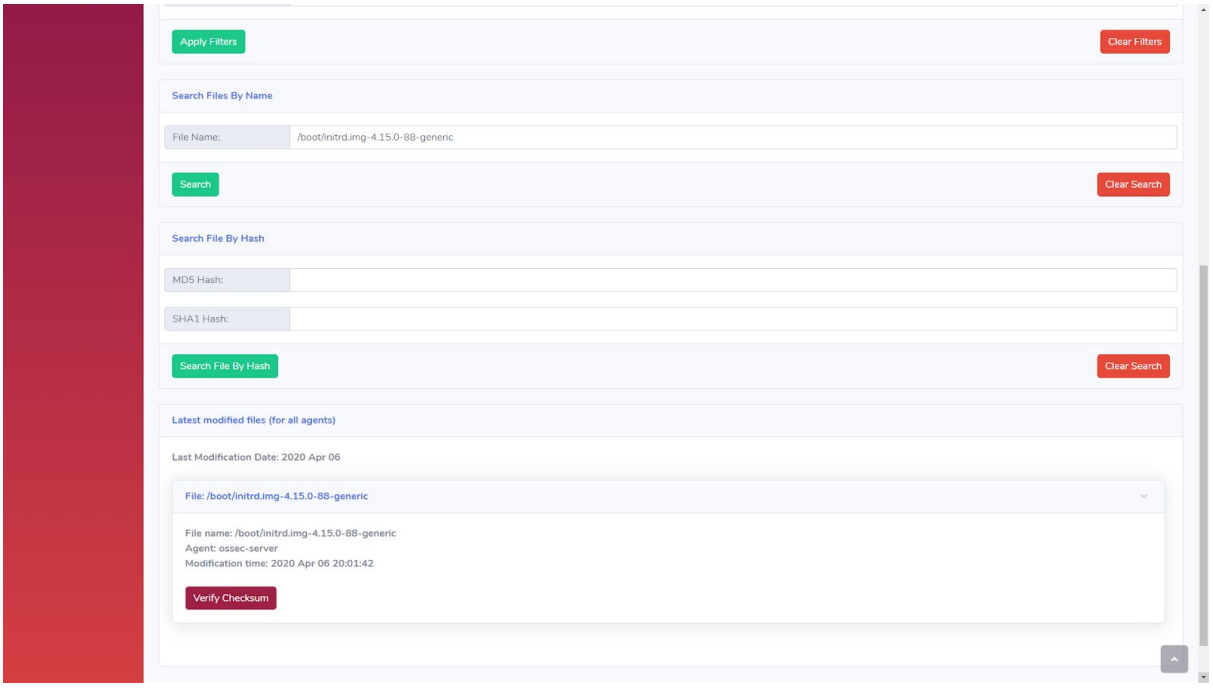

Figura 4.2.6 Filtro de búsqueda por nombre en Integrity Checking

Añadido a estos filtros, hemos desarrollado el código para incorporar un buscador de archivos por nombre, o directorio completo, que nos devuelve una lista de archivos cuyo nombre tenga alguna similaridad con lo buscado, como podemos ver en la Figura 4.2.6. Si estos archivos no se han visto comprometidos, o no ha habido ninguna coincidencia, no se mostrarán datos. De forma análoga hemos añadido al buscador la posibilidad de buscar la función resumen (*hash*) en formato SHA1 o MD5 de un archivo, y tendrá un resultado parecido al de búsqueda por nombre: nos muestra el archivo que tiene ese *hash*.

Un cambio muy importante con respecto a la WUI original que hemos implementado es la eliminación del botón "Dump Database". Hemos integrado los datos que se obtenían con esta funcionalidad en la misma vista, y estarán ocultos hasta que el usuario haga click en el botón "Verify Checksum", haciendo que aparezca un modal como el que mostramos a continuación en la figura 4.2.7.

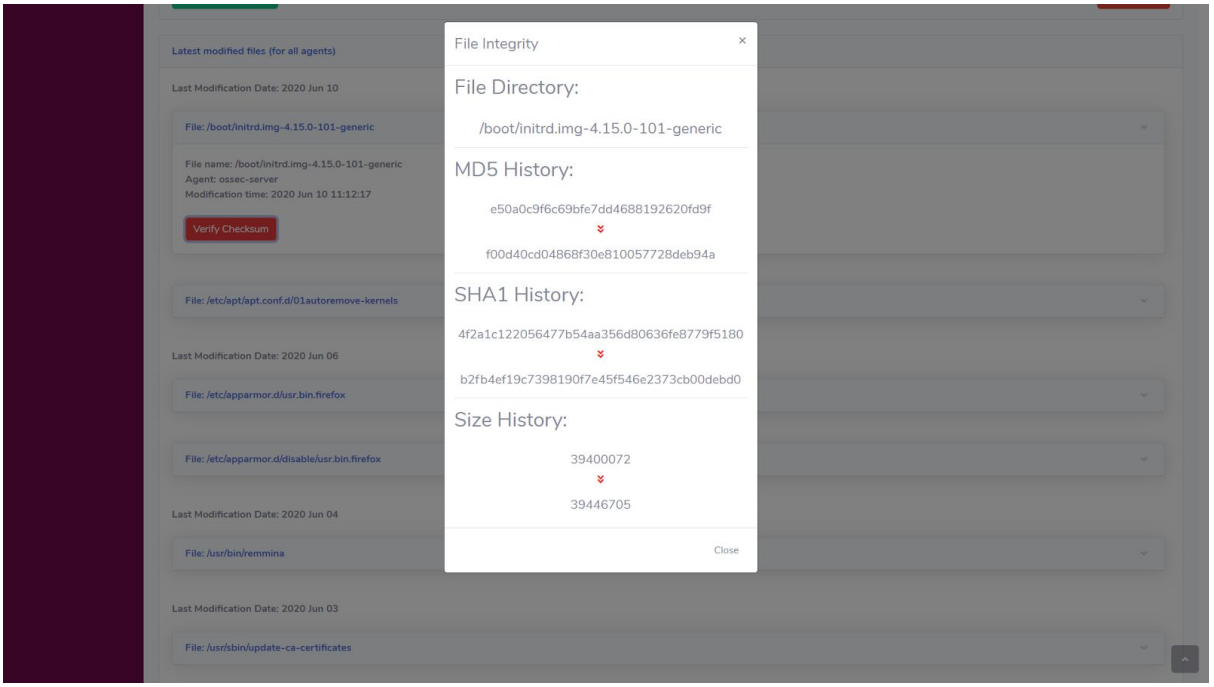

Figura 4.2.7 Modal con la evolución de la integridad

El contenido del modal nos muestra la evolución de la integridad (los cambios que han surgido en el *hash* del archivo) y el tamaño del archivo en concreto.

Este flujo de acciones para un usuario resulta más cómodo e intuitivo que el original, ya que nos agrupa en la misma vista la información de la alerta y la integridad del archivo. Podemos acceder a toda la información de manera lineal, y sin tener que bajar hasta otra sección para ver más información.

Originalmente, esta pestaña no mostraba realmente la integridad de los datos, sino solo las alertas, a no ser que hiciésemos click en “Dump Database”, añadiendo más pasos de los necesarios a un proceso que debería ser claro y conciso.

4.2.4 Stats

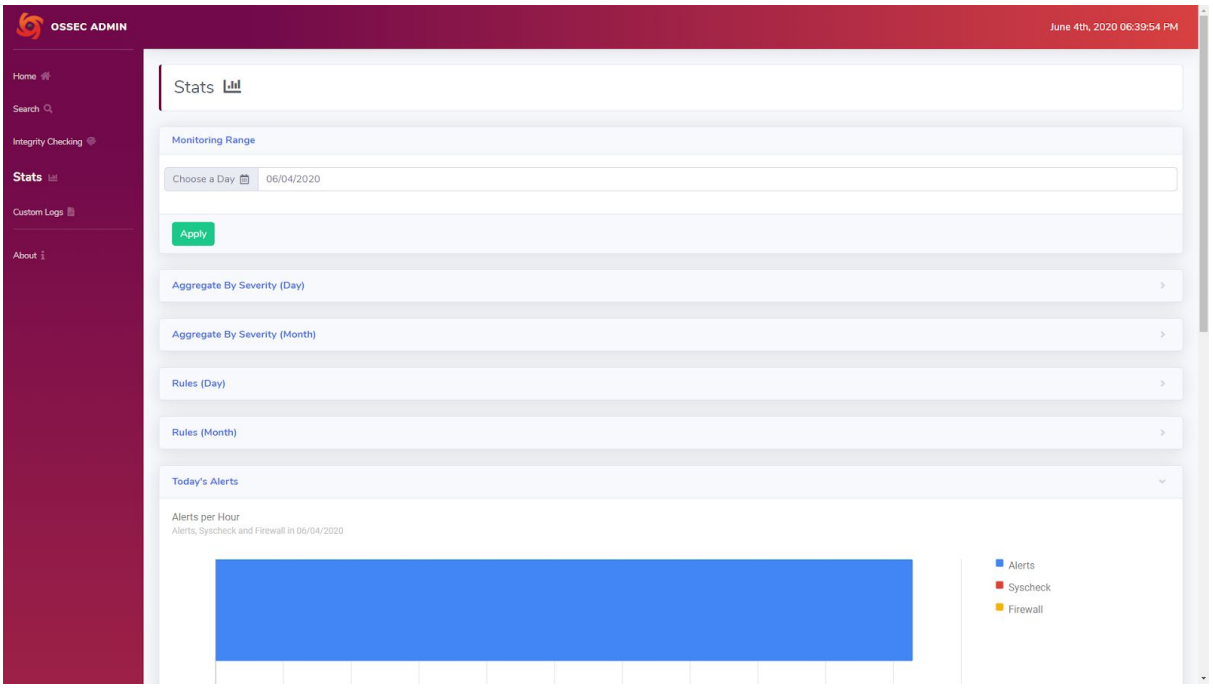

Figura 4.2.8 Página Stats de la WUI implementada

Stats es la vista (Figura 4.2.8) que más dista de la original (Figura 4.1.6), que mostraba los datos estadísticos en meras tablas sin formato. Hemos hecho uso de librerías de código abierto como Google Charts y DataTables. Como hay mucha información en esta vista, vamos a analizar cada sección por separado.

Primero de todo, tenemos un selector de fecha que nos permite acotar el rango de análisis de los datos que nos muestran las gráficas diarias. Hemos visto oportuno que siempre que se elija un día, le agregamos también las estadísticas de ese mes, puesto que anteriormente se hacía de forma separada, y te exigía tener que abrir más de una ventana para ver una comparativa. Hemos realizado esta opción para que el usuario pueda comparar las alertas del día con la mensual y hacerse una idea de las anomalías que pudiesen existir respecto al mes, o visualizar si ese día compone la mayor parte de alertas del mes.

Parte del trabajo implementado ha consistido en dotar a la WUI de una serie de gráficas que faciliten la visualización de los eventos que se han producido, de tal forma que cualquier usuario pueda interpretar de manera sencilla la gravedad de los acontecimientos registrados.

La primera gráfica “Aggregate by Severity (Day)” nos muestra los datos agrupados por el grado de gravedad de las alertas que se han encontrado en el día seleccionado. El degradado de los colores hace que visualmente se perciba mejor la gravedad de la alerta, acompañado de los *labels* de cada nivel en el eje de las abscisas. Si movemos el cursor hacia cualquiera de las barras, podemos ver cuántas alertas de ese nivel han sido lanzadas (figura 4.2.9).

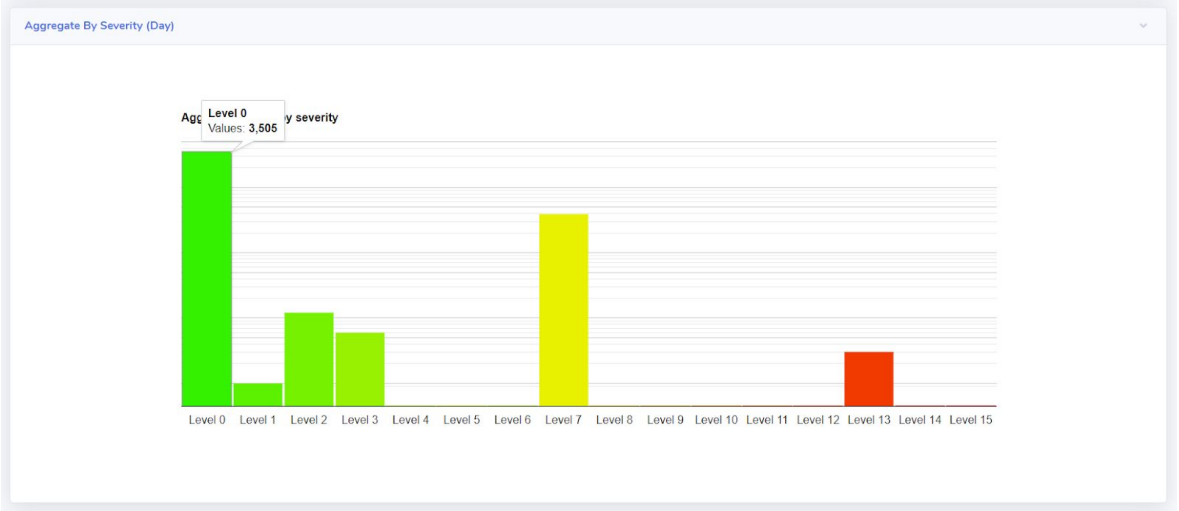

Figura 4.2.9 Estadísticas en formato columnas

La segunda gráfica con el nombre de "Aggregate by Severity (Month)", tiene el mismo formato que la primera gráfica, pero contiene la acumulación de datos recogidos a lo largo de todos los días del mes actual (o del mes del día elegido).

A continuación nos encontramos con dos tablas que enumeran el número de alertas por cada regla, así como el porcentaje del total que representa cada cada una de ellas: la primera tabla (figura 4.2.10) recogiendo los datos del día elegido, y la segunda nuevamente los datos del mes.

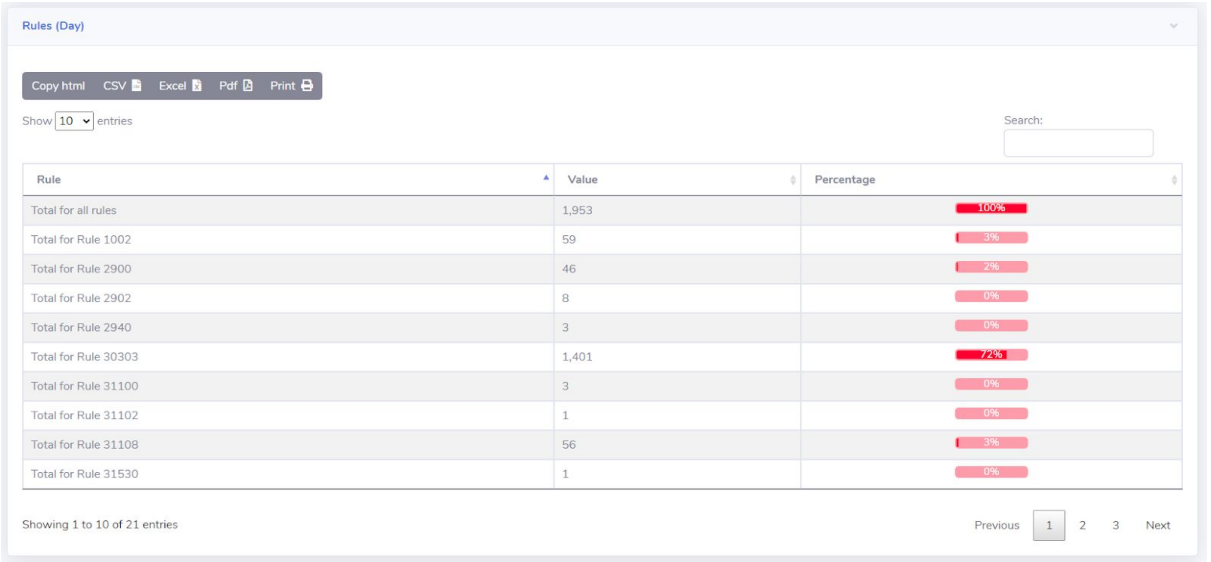

Figura 4.2.10 Estadísticas de las reglas en formato tabla

Encima de cada tabla, nos encontramos con una barra de herramientas que nos permite exportar los datos de la tabla en distintos formatos, como es en HTML, archivo CSV, archivo Excel, PDF o formato listo para imprimir. Ya que es frecuente que haya una gran cantidad de datos, también podemos visualizar las entradas divididas en páginas.

Por último, tenemos dos gráficas que nos muestran el historial de alertas por hora del día actual, como muestra la figura 4.2.11, y el resumen mensual (figura 4.2.12):

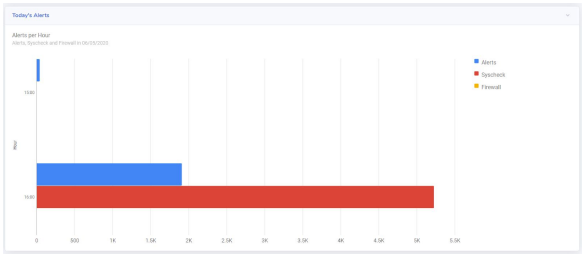

Figura 4.2.11 Estadísticas del historial

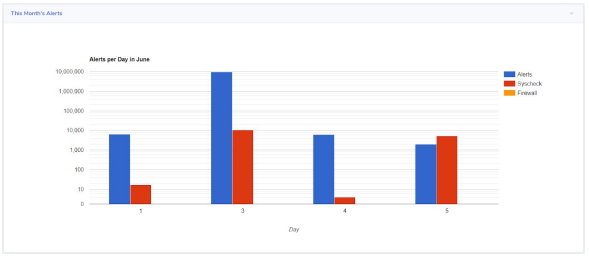

Figura 4.2.12 Estadísticas del resumen mensual

En general, esta vista intenta mostrar la misma información que nos mostraba la WUI original, pero haciendo uso de bibliotecas presentes en casi cualquier aplicación web, como son las gráficas, en nuestro caso usando la librería de Google Charts, y las tablas con formato enriquecido de DataTables.

4.2.5 Custom Logs

Esta pestaña nos ofrece una nueva funcionalidad que no presentaba la WUI oficial del HIDS. OSSEC ofrece la oportunidad de poder introducir nuevas monitorizaciones con una configuración específica, pero para ello debes introducir manualmente en el fichero ossec.conf o ejecutar el script util.sh (con muy poca variedad de configuración). Por ello creamos un formulario con varias opciones de configuración. La figura 4.2.13 muestra qué aparece cuando seleccionamos esta pestaña. La primera opción es escoger el tipo de monitorización que queremos realizar, si de tipo comando, fichero, DNS o Website.

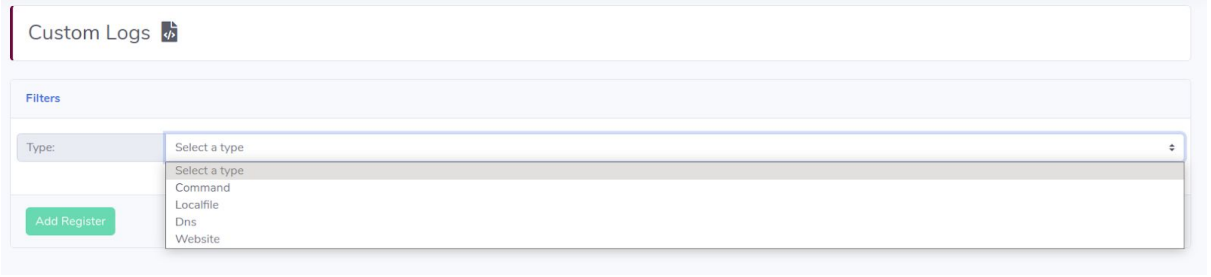

Figura 4.2.13 Página Custom Logs de la WUI implementada

En la opción *Command,* figura 4.2.14, nos despliega los principales atributos con los que puede configurarse un comando. Además ofrecemos la posibilidad de añadir la respuesta activa a este comando

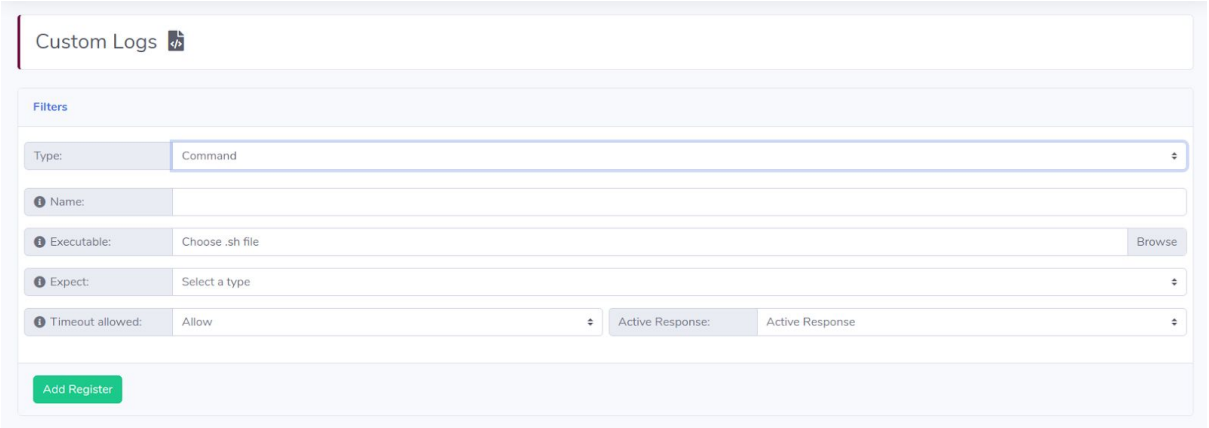

Figura 4.2.14 Elección tipo Command en Custom Logs

Si el usuario indica que si quiere *Active Response* se despliega otro apartado más, como podemos observar en la figura 4.3.15, donde el usuario introduce la configuración deseada al responsive. Si vuelve a indicar que no, de vuelve a ocultar este.

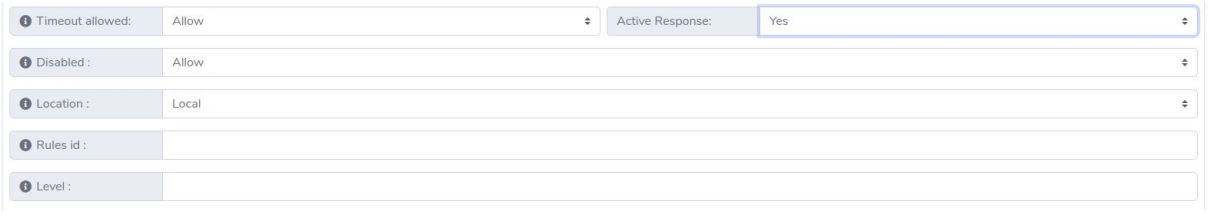

Figura 4.2.15 Elección de agregar Active Response

Si la opción es Localfile, nos encontramos dos posibles casos: Command o File. Estos se diferencian en que el primero es para monitorizar un comando ya existente y el segundo un fichero de tipo log que exista en el host. En algunos atributos solo te permiten elegir entre unos ciertos valores que aparecen en forma de lista. En la figura 4.2.16 podemos observar la opción de añadir un fichero.

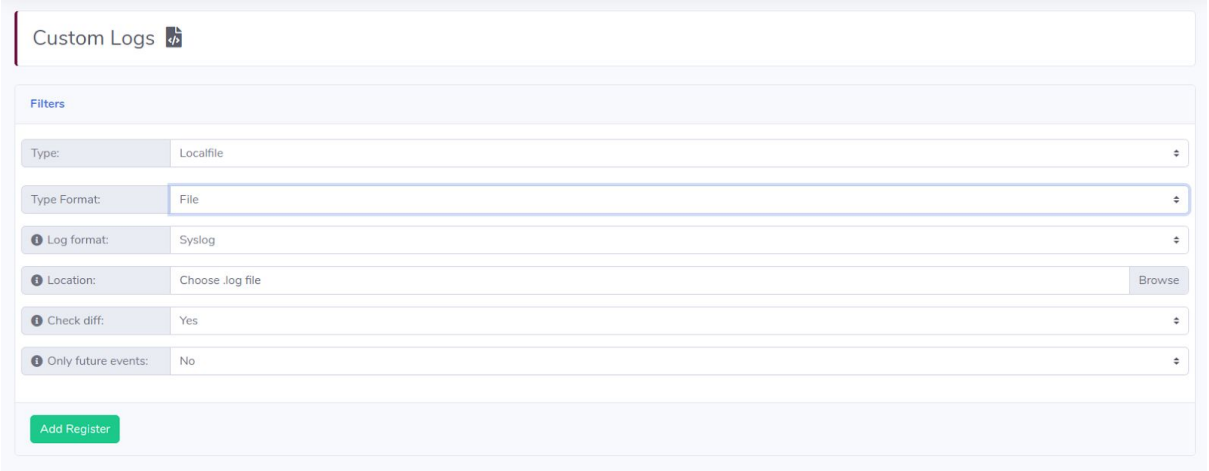

Figura 4.2.16 Elección tipo Localfile en Custom Logs

Las dos últimas opciones son la monitorización de *DNS* y *Website*. Tienen en común que las dos opciones sólo tienen un solo input para añadir. El primero una dirección IP y en la segunda una dirección web. Aquí mostramos un ejemplo en la Figura 4.2.17.

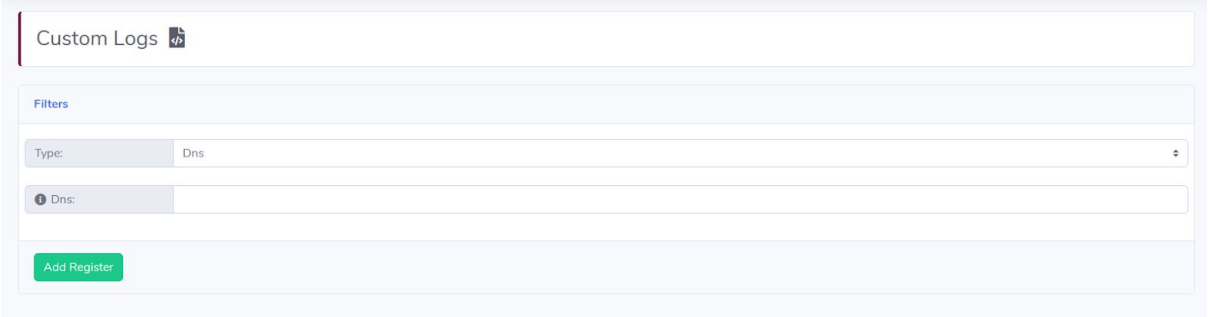

Figura 4.2.17 Elección tipo DNS en Custom Log

4.2.6 About y Error 404

Aunque son dos vistas sin información relevante, aportan cierta formalidad a la estructura final de la aplicación. La página de *About* (figura 4.2.18) contiene un pequeño disclaimer, y la página 404 (Figura 4.2.19) es por defecto la página de redirección en caso de que se haga una petición inesperada al servidor con una ruta incorrecta, con una sutil animación en el mensaje de error.

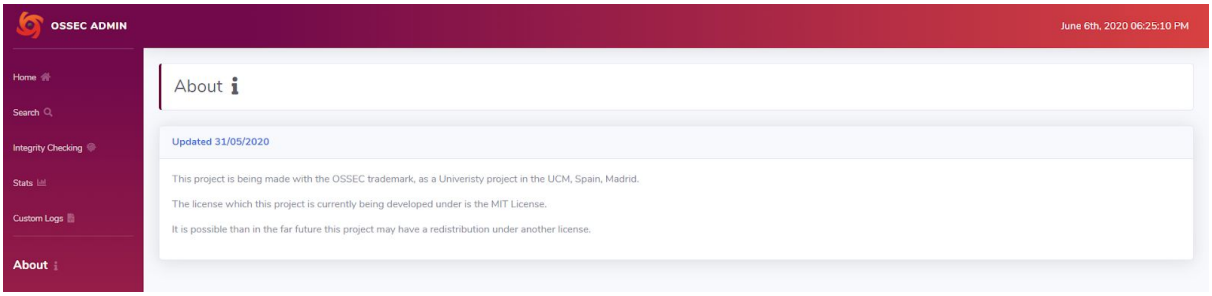

Figura 4.2.18 Página About de la WUI implementada

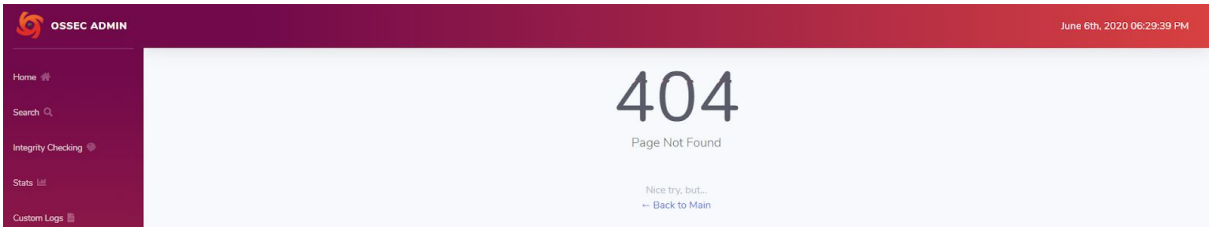

Figura 4.2.19 Error 404

4.3 Especificaciones Técnicas Front

Para entender mejor el desarrollo del frontend de la nueva WUI, analizaremos aquellos elementos más relevantes que se repiten a lo largo de la aplicación.

La plantilla general reutilizada a lo largo de la página utiliza la función *require(...)* que incluyen los archivos comunes a todas las vistas, como son la barra lateral, la cabecera y el pie de página, además de importar el archivo *config.php* que controla la configuración general de la aplicación.

En el tag <*head*> incluimos aquellos scripts específicos de la vista actual, a la vez que aquellos compartidos por todas las vistas (recogido en el archivo *allCSS.php*), como son los archivos css de estilo.

```
<?php require('includes/config.php'); ?>
...
<head>
          <?php require('imports/allCSS.php'); ?>
          ...
</head>
```

El cuerpo o body cuenta con un identificador que nos permite hacerle referencia para ascender en la página en cualquier momento gracias a un botón que encontramos cuando nos desplazamos con la barra lateral de la página. El contenido de este div es el contenido de toda la vista, y tiene la siguiente estructura:

```
<body id="page-top">
  <div id="wrapper"> <!-- Page Wrapper -->
    <?php require('includes/sidePanel.php'); ?> <!-- Sidebar -->
    <div id="content-wrapper" class="d-flex flex-column">
      <div id="content"> <!-- Main Content -->
        <?php require('includes/topBar.php'); ?> <!-- Topbar -->
        <div class="row">
        ...
        </div>
      </div> <!-- End of Main Content -->
      <?php require('includes/footer.php'); ?> <!-- Footer -->
    </div> <!-- End of Main Content -->
  </div> <!-- End of Page Wrapper -->
  <!-- Scroll to Top Button-->
  <a class="scroll-to-top rounded" href="#page-top">
    <i class="fas fa-angle-up"></i>
  </a>
  <!-- End of Scroll Button -->
```

```
</body>
```

Como podemos observar, dentro del contenido principal hacemos uso de un div de clase row, propio de Bootstrap, los cuales son el principal elemento que dota a una página de *responsiveness*, lo cual hace que la vista se adapte a cualquier tipo de pantalla, independientemente de de su tamaño o resolución. En general, siempre que queremos dividir contenido en horizontal, insertamos ese contenido en una fila (row), y si queremos dividir una fila en varias columnas, tenemos que hacer uso de columnas (col).

La regla principal en la que se basa Bootstrap es que cuando definimos un elemento de tipo row, habrá un tamaño por defecto de 12 para repartir a lo largo de la fila. Este tamaño no define un número de píxeles, sino que es una referencia a tener en cuenta siempre que añadimos columnas dentro de una fila. Por ejemplo, en una fila pueden caber 4 columnas de tamaño 3 o 3 columnas de tamaño 4, 2 columnas de tamaño 6, etc. Si a su vez, añadimos en una columna de tamaño X otra fila, dentro de esa fila, volveremos a tener de referencia el 12 para dividir esa fila nuevamente. Fijándonos en la figura 4.3.1, podemos ver una representación gráfica de lo que hemos explicado anteriormente.

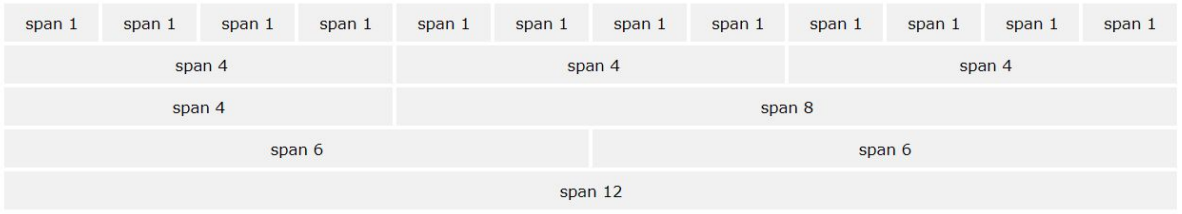

Figura 4.3.1 Display de filas y columnas en Bootstrap

Esto permite que no tengamos que definir para cada elemento su posicionamiento de manera individual, lo cual siempre ha sido tedioso en el desarrollo de páginas web. Bootstrap en definitiva consigue unificar los elementos de posicionamiento en clases para que su uso sea más intuitivo.

Al final de cada página, hacemos llamadas a los scripts de JavaScript que utilicemos para esa vista, y aquellos scripts que se utilicen en común para todas las vistas, como Bootstrap.

```
...
</html>
<?php include('imports/all.php'); ?>
<script>...</script>
```

El archivo all.php recoge todas las sentencias include de los scripts de JavaScript comunes a la página. En algunos casos, los cuales mencionaremos más adelante, en los que necesitamos usar variables de PHP en los scripts, estas funciones aparecerán dentro del apartado <script>. No es la

mejor decisión que se puede tomar, ya que es mejor práctica externalizar las funciones en archivos separados, pero nos hemos visto obligados por simplificar el desarrollo.

A lo largo de toda la WUI, encontramos una estructura que se repite en todas las vistas, la cual es denominada *card*. Esta estructura permite dividir de manera clara cada elemento y le aporta riqueza visual, pudiendo añadir una cabecera, un cuerpo, y un pie (para incluir botones de *Submit* y *Cancel*, al elemento en concreto. Si usamos el elemento como envoltorio de un formulario, encontramos algo como en la figura 4.3.2.

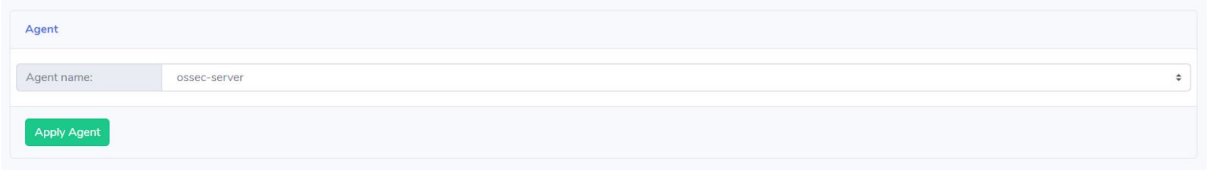

Figura 4.3.2 Formulario para cambiar de Agente en Integrity Check

Es un formato simple, pero elegante, y permite modificar el contenido interno sin preocuparse por los elementos estáticos, como son la cabecera y el pie. Además de esto, en muchos casos, hemos añadido la funcionalidad *collapse* al elemento card para que podamos desplegar la información contenida cuando queramos, o cerrar si ocupa mucho espacio y nos queremos olvidar de ello, como podemos ver en las figuras 4.3.3 y 4.3.4.

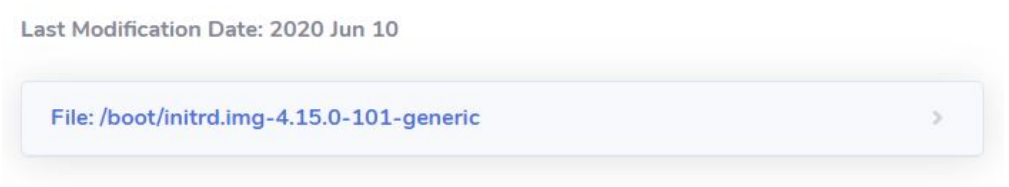

Figura 4.3.3 Elemento card plegado

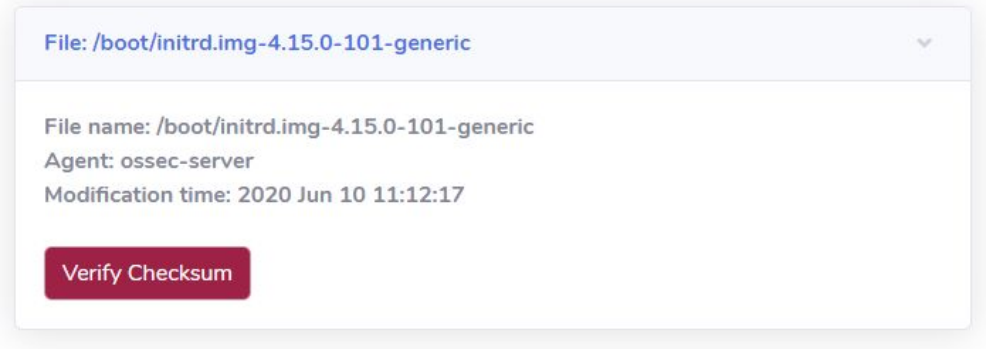

Figura 4.3.4 Elemento card desplegado

La barra lateral está compuesta por un elemento <ul> o "lista desordenada" que contiene elementos <li> "list item" con anclas <a> que permiten navegar a todas las vistas de la aplicación desde cualquier vista. Ya que el diseño original de la librería que utilizamos no nos convencía, decidimos alterar los colores de la barra, así como el comportamiento de los elementos ancla

cuando hacemos click en ellos o pasamos por encima. Esto nos permitirá diferenciar claramente en qué vista nos encontramos en todo momento. Si hacemos click en el logo de OSSEC, volveremos a la página inicial.

```
<ul class="navbar-nav bg-gradient-primary sidebar sidebar-dark accordion" id="accordionSidebar">
<!-- Sidebar - Brand -->
          <a class="sidebar-brand d-flex align-items-center justify-content-center" href="index.php">
                    <div class="sidebar-brand-icon rotate-n-15">
                              <img src="img/ossec.png" width="50px" height="50px">
                    </div>
                    <div class="sidebar-brand-text mx-2">
                              OSSEC Admin
                    </div>
          </a>
          ...
          <!-- Nav Item -->
          <li class="nav-item">
          <a class="nav-link" href="index.php" aria-expanded="true" aria-controls="collapseTwo">
          <?php
            if ($page == "main") {
              echo('<span class="nav-text-custom-active">');
            } else {
              echo('<span>');
            }
          ?>
          Home  </span><i class="fas fa-home"></i>
          </a>
          </li>
          ...
</ul>
```

Para destacar el nombre de la vista en la que nos encontramos, comprobamos el valor de la variable "$page" la cual asignamos al inicio de la vista en concreto. Dependiendo del valor de esta variable, le asignamos la clase propia que hemos creado para el elemento seleccionado.

```
.sidebar .nav-item .nav-link span:hover {
    color: white;
    font-weight: bold;
    font-size: large;
```

```
}


.sidebar .nav-item .nav-link span.nav-text-custom-active {
    color: white;
    font-weight: bold;
    font-size: larger;
}
```

4.3.1 Funciones Específicas de Integrity Checking

Aunque ya hemos hablado del funcionamiento en general de la vista *Integrity Checking* a la hora de hablar de las mejoras del frontend, creemos que es oportuno entrar en detalle en el funcionamiento de la representación de los datos y en el modal que podemos abrir para verificar la integridad de los archivos.

Primero de todo, es importante explicar la decisión que se ha tomado a la hora de dividir la sección superior en cuatro peticiones distintas. El primer formulario, nos deja modificar el agente sobre el cual se están analizando los datos, mientras que los restantes son utilizados para filtrar los datos que recuperamos. Todos los formularios son representados como elemento card, y cuando hacemos submit sobre un formulario, se accede a las peticiones backend de forma distinta. Es decir, un formulario excluye los resultados del otro. Esto es debido a la forma en la que se recuperan los datos, que se hace desde dos secciones distintas, y una depende de la otra.

Si escogemos filtrar por hash de archivo, se excluirán los filtros básicos y el filtrado por nombre, ya que es un filtrado muy específico que en el gran porcentaje de casos solo va a devolver un resultado, o ninguno. De manera homóloga sucede cuando filtramos por nombre de archivo, puesto que necesitamos excluir los filtros anteriores para poder acceder a todo el registro de archivos.

Cuando queramos vaciar los filtros, valdría con hacer click en cualquiera de los botones de “Clear” de los formularios, puesto que los tres están ligados a la misma función “onclick”:

```
function clearFilters() {
    var maxDays = document.getElementById('maxDays');
    maxDays.value = 100;
    var maxFiles = document.getElementById('maxFiles');
    maxFiles.value = 100;
    var dayOrder = document.getElementById('dayOrder');
    dayOrder.selectedIndex = 1;
    var fileOrder = document.getElementById('fileOrder');
    fileOrder.selectedIndex = 1;
```

```
    var md5 = document.getElementById('md5');
    md5.value = "";
    var sha1 = document.getElementById('sha1Hash');
    sha1.value = "";
    var fileName = document.getElementById('fileName');
    fileName.value = "";
}
```

Para reutilizar el código incluido en la plantilla HTML, hemos ocultado la visibilidad del modal que será mostrado al hacer click sobre *Verify Checksum*, ya que lo usaremos para poblarlo con los datos específicos del archivo elegido.

```
<div class="modal fade" id="myModal" role="dialog">
      <div class="modal-dialog">
        <!-- Modal content-->
        <div class="modal-content">
          <div class="modal-header" style="display: block;" >
            <button type="button" class="close" data-dismiss="modal">&times;</button>
            <h4 class="modal-title">File Integrity</h4>
          </div>
          <div class="modal-body">
            <div class="" id="nof"></div>
            <hr>
            <div class="" id="md5"></div>
            <hr>
            <div class="" id="sha1"></div>
            <hr>
            <div class="" id="size"></div>
          </div>
          <div class="modal-footer">
            <button type="button" class="btn btn-default" data-dismiss="modal">Close</button>
          </div>
        </div>
      </div>
</div>
```

Al hacer click en el botón de *Verify Checksum*, se llama a una función JavaScript encargada de poblar el modal con los datos específicos del archivo elegido. En vez de modificar por completo la

forma de recuperar los datos del backend, en este caso nos hemos amoldado al formato original, y hemos parseado cada dato para agruparlos para la visualización.

```
function loadSpecificChecksum(element) {
    var path = element.getAttribute('fileName');
    var array = <?php  echo json_encode($db_changes);?>;
    var name = ""; var md5 = "";
...
```

```
for (var i = 0; i < array.length; i++) {
      if (array[i][i].name == path) {
        name = array[i][i].name;
        md5 = array[i][i].sum;
        size = array[i][i].size;
        break;
      }
}
md5 = md5.split('->');
size = size.split('->');
var allMD5 = [];
var allSHA1 = [];
var allSizes = size;
for (var i = 0; i < md5.length; i++) {
      var fila = md5[i].split(' ');
      allMD5.push(fila[1]);
      allSHA1.push(fila[3]);
}
...
```

Tras almacenar los datos en estructuras propias de JavaScript, accedemos a cada elemento de la plantilla y sobreescribimos la información a mostrar.

```
var icon = '<p style="color: red; text-align: center;"><i class="fas fa-angle-double-down"></i></p>';
var nof = document.getElementById('nof');
nof.innerHTML = "<h2>File Directory:</h2><br>"
     + "<h4 style='text-align: center;'>" + name + "</h4>";
var md5Div = document.getElementById('md5');
var html = "<h2>MD5 History:</h2><br>";
```

```
for (var i = 0; i < allMD5.length; i++) {
      html += '<h5 style="text-align: center;">' + allMD5[i] + '</h5>';
      if (i != allMD5.length - 1) {
        html += icon;
      }
}
...
```

4.3.2 Funciones Específicas de Stats

Por último, vemos necesario destacar la programación de las funcionalidades que hemos añadido a la vista de *Stats*, ya que usamos librerías externas que requieren de llamadas a una API (como en el caso de Google Charts) o de funciones propias de la librería.

Empezando por las gráficas de Google Charts, el código siempre tiene la misma estructura:

```
google.charts.load('current', {'packages':['bar']});
google.charts.setOnLoadCallback(drawDailyChart);
```

Primero de todo se hace una llamada a la API de Google Charts, especificando el tipo de visualización y el paquete de gráficas a implementar. Una vez recibimos la respuesta de la API (con la función *google.charts.setOnLoadCallback(drawDailyChart))* llamamos a la función que contiene el código que generará la visualización de la gráfica.

Dentro de la función, el primer paso consiste en recoger los datos que hemos obtenido del back (que hemos modificado para que podamos utilizarlos como mejor nos convenga). Hemos decidido incluir las funciones de PHP en el script de JavaScript (aunque hemos externalizado las funciones a un archivo separado), ya que es más cómodo utilizar las variables que hemos inicializado en nuestro archivo *initStats* en vez de hacer una petición AJAX y recuperar cada grupo de datos por separado.

```
function drawDailyChart() {
  var hours = [];
  var alerts = [];
  var syschek = [];
  var firewall = [];
  <?php
  for ($i = 0; $i < 23; $i++) {
    ?>
    var aux = [<?php echo $daily_stats{'alerts_by_hour'}[$i]; ?>];
```

```
    if (aux.length > 0) {
      var hour = '' + <?php echo $i; ?> + ':00'
      hours.push(hour);
      alerts.push(aux[0]);
    }
...
```

Una vez hemos conseguido formar arrays en JavaScript poblados con los datos que teníamos en las variables de PHP, procedemos a configurar los datos que requiere la gráfica:

```
var tabla = [['Hour', 'Alerts', 'Syscheck', 'Firewall']];
  for (var i = 0; i < hours.length; i++) {
    var fila = [];
    fila.push(hours[i]);
    fila.push(alerts[i]);
    fila.push(syschek[i]);
    fila.push(firewall[i]);
    tabla.push(fila);
  }
```

Formamos un array de arrays de forma que la primera entrada sea la cabecera de la gráfica (la leyenda formada por los nombres de cada una de las variables), y las demás entradas serán los datos que hemos recuperado. Una vez hecho esto, convertimos estos datos en datos renderizados para que los interprete la API de Google:

```
  var data = google.visualization.arrayToDataTable(
    tabla
  );
```

Finalmente, rellenamos las opciones de la estructura siguiendo las pautas del tipo de gráfica en concreto, y mandamos la petición final con todos los datos renderizado al generados de objetos de tipo Charts. Una vez nos devuelve el objeto, procedemos a su representación visual, especificando la sección del documento donde queremos representarla.

```
var options = {
    chart: { title: 'Alerts per Hour',
             subtitle: 'Alerts, Syscheck and Firewall in ' + date,
    },
    width: 1400, height: 600,
    hAxis : {
```

```
      textStyle : {
        fontSize: 15
      },
      showTextEvery: true,
    },
    bars: 'horizontal' // Required for Material Bar Charts.
};
```

```
  var chart = new google.charts.Bar(document.getElementById('dayChart'));
  chart.draw(data, google.charts.Bar.convertOptions(options));
```

Éstos son los pasos que hemos seguido cada vez que queríamos generar una gráfica nueva, sea del tipo que sea. Para las tablas en cambio, el código se simplifica, puesto que la librería de DataTables es menos soficsticada y se apoya en una tabla ya formada previamente sin estilizar. Cuenta con muchas opciones (como paginación, barra de búsqueda, toolbar de exportación, etc. y permite aportar mayor riqueza a la representación visual.

Al introducir en la plantilla HTML los datos generales de nuestras tablas, procedemos a llamar a la siguiente función, la cual va a formatear la tabla conforme especifiquemos en la estructura de datos:

```
var table = $('#rules_table').DataTable({
    "dom": '<B><"row" lf>rt<"my-4" ip>',
    "paging": true,
    "colReorder":true,
    "autoWidth": true,
    "columnDefs": [{
          targets: 2,
          render: $.fn.dataTable.render.percentBar('round','#fff', '#FF9CAB', '#FF0033', '#FF9CAB', 0,
'solid')
    }],
    "buttons": [
                    { "extend":'copyHtml5', "text": 'Copy html',"className":'btn btn-secondary'},
                    {  "extend":'csvHtml5',  "text":  'CSV  <i  style="margin-left:4px;"  class="fas
fa-file-csv"></i>',"className":'btn btn-secondary'},
                    {  "extend":'excelHtml5',  "text":'Excel  <i  style="margin-left:4px;"  class="fas
fa-file-excel"></i>',"className":'btn btn-secondary'},
                    {   "extend":'pdfHtml5',   "text":'Pdf   <i   style="margin-left:4px;"   class="far
fa-file-pdf"></i>',"className":'btn btn-secondary'},
                    {   "extend":'print',   "text":'Print   <i   style="margin-left:4px;"   class="fas
fa-print"></i>',"className":'btn btn-secondary'},
```

```
    ]});
```

Como podemos ver, hemos definido algunos de los parámetros de las columnas y de los botones de la barra de herramientas que nos permitirá exportar los datos a cualquiera de los formatos que elijamos.
Para especificar la clase para los botones y las funciones de la tabla, tenemos que incluir algunas características de tipado como vemos a continuación:

```
var buttons = $("#rules_table_wrapper .dt-buttons");
buttons.addClass("btn-group my-3");
var filter= document.querySelector("#rules_table_filter");
filter.className= "input-group col-sm-2";
filter.children[0].children[0].className="form-control";
var show= document.querySelector("#rules_table_length");
show.className= "col-sm-10";
```

Tras estos pasos, hemos completado la función de JavaScript y podremos visualizar la tabla como muestra en la figura 4.3.5.

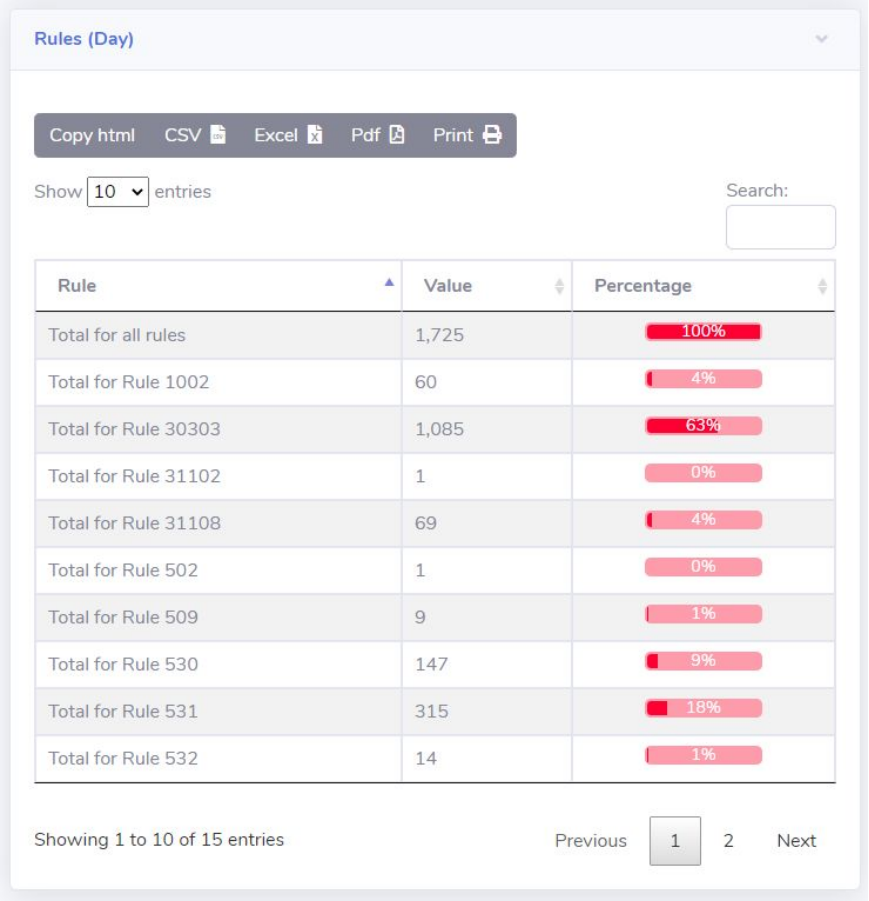

Figura 4.3.5 DataTable para reglas del día actual en Stats

4.3.3 Funciones Específicas de Custom Logs

En la vista *Custom Logs*, utilizamos funciones de JavaScript para hacer visibles más o menos secciones, dependiendo de la selección anterior. Al elegir el tipo *command*, por ejemplo, se hacen visibles las opciones siguientes, como podemos ver en las figuras 4.3.6 y 4.3.7.

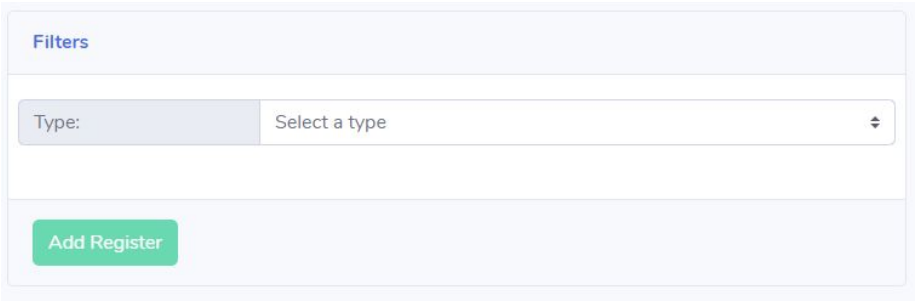

Figura 4.3.6 Formulario inicial Custom Logs

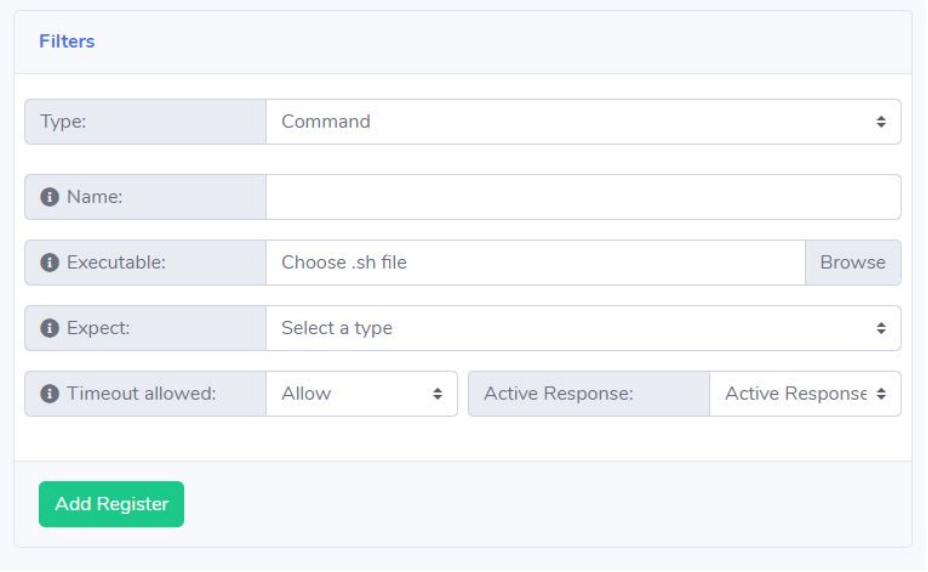

Figura 4.3.7 Formulario Custom Logs Command

Dado que el código es muy largo, vamos a analizar un caso de uso en concreto. Partiendo de la figura 4.3.7, y eligiendo la opción *command*, asignando un atributo *onchange* al selector:

```
function changeType(element) {
          var htmlSection = document.getElementById('typeContent');
          document.getElementById('secondLocalFile').innerHTML = "";
          htmlSection.innerHTML = "";
          document.getElementById('submitRules').disabled = false;
          switch (element.options[element.selectedIndex].value) {
                    case 'command':
```

```
                                generateCommandHTML(htmlSection);
                        break;
            ...


function generateCommandHTML(element) {
            html = `
            <div class="input-group mb-3">
                        <div class="input-group-prepend">
                        <label style="width: 200px;" class="input-group-text" for="inputGroupSelect01">
                        <i    class="fas    fa-info-circle"    title="`    +    tags.name.info    +    `"
style="margin-right:6px;"></i>
                        ` + tags.name.name + `:
                        </label>
            </div>
            <input required id="name" type="text" class="form-control" name="` + tags.name.ossec_name +
`" value="">
            </div>
...
            element.innerHTML = html;
}
```

La función *changeType(element)* recibe el elemento <select> como parámetro. Se extrae la opción elegida y se localiza la sección HTML donde irá insertado el resto del formulario. Se comprueba el tipo de opción seleccionada mediante un *switch* y se llama a la función encargada de generar y asignar el HTML renderizado a la sección del formulario.

Hay maneras más elegantes de ir generando secciones de un formulario, pero tomamos esta decisión con intenciones de mejorarla en caso de que hubiese tiempo. Una idea que aún tenemos en mente es en dividir cada subsección del formulario en archivos distintos, que se vayan solapando a medida que se elige una opción u otra.

5. OSSEC WUI - Back

La aplicación original de OSSEC es muy completa y, aunque su aspecto visual no encaja con los estándares de una aplicación moderna, su robustez "interna" es incuestionable. Sin embargo, a lo largo de todo el código se repite un problema de diseño que hace que poder mejorar la herramienta se dificulta, ya que no hay una clara diferenciación entre las capas de front y el back, por ejemplo, muchas veces se mezclan llamadas al sistema operativo con el código HTML que vemos.

En definitiva, es en la arquitectura donde realmente la herramienta es más débil . Si se hubiese diseñado con una clara diferenciación entre las capas de front y back, el trabajo presentado en esta memoria hubiera consistido únicamente en reformar la representación de los datos. Para que todo el desarrollo presentado en el capítulo 4 pueda funcionar eficientemente y haya sido factible realizar esos cambios nos hemos visto obligados a tener que modificar varios archivos para separar las acciones del *backend* de las del *frontend*. A la vez que realizábamos este trabajo hemos tenido que modificar más aspectos, hasta incluso cambiar el formato en el que la herramienta recogía los datos de monitorización.

5.1 Análisis del Backend

Para entender el back, vamos a realizar un pequeño análisis de los archivos más importantes que tiene la WUI. Estos ficheros mezclan parte del front y parte del back, las funciones del back ya devuelven HTML, por lo que su análisis es complejo. Para poder analizarlos y presentarlos en esta memoria de una manera ordenada, dividiremos los siguientes apartados según las vistas.

5.1.1 Index.php

Es el fichero que controla todas las vistas, y que cambia todo su contenido según el parámetro que le llega por la URL (vía GET). Esta era una de las partes de la herramienta que consideramos que no estaba programada de una forma eficiente , puesto que obligaba a las funciones de las distintas vistas a devolver formato HTML, en vez de devolver estructuras con la información requerida. Estas funciones no se iban a poder reutilizar en otras partes del código si se necesitase, y además añade la dificultad de comprensión al código.

Los argumentos que le pueden llegar a esta vista a través de la variable 'f' ($GET['f']) son:

- $GET['f'] = 's' : Muestra la vista de búsqueda (site/search.php)
- $GET['f'] = 'm' : Muestra la vista principal (site/main.php)
- $GET['f'] = 't' : Muestra la vista de estadísticas de alertas (site/stats.php)

- $GET['f'] = 'i' : Muestra la vista de chequeo de integridad (site/syscheck.php)
- $GET['f'] = 'a' : Muestra un información sobre la herramienta (site/help.php)

5.1.2 Main.php

La vista principal de OSSEC es *Main*, en la cual podemos ver los agentes disponibles, los últimos ficheros modificados y los últimos eventos.

Antes de entrar en detalle sobre las funciones que encontramos en esta vista (muchas de las funciones comunes a todas las vistas están agrupadas dentro del archivo Ossec_handle.php), vamos a explicar brevemente cuales son dichas funciones y cuál es su utilidad:

Ossec_handle.php: Tiene dos funciones principales: os_check_config y os_handle_start:

- *os_handle_start()*: Crea un directorio (DIR/queue/agent_info) y un objeto manejador (handler) en caso de no estar ya creado.
- *os_check_config()*: Comprueba que la configuración de **ossec_conf.php** es correcta. Este fichero se encarga de declarar las variables, indicar el número máximo de alertas, el tiempo de refresco y el tiempo de búsqueda máximo. Esta función recibe como parámetro la dirección del directorio, el número máximo de alertas por página, el nivel de alertas, el tiempo de búsqueda y el tiempo de refresco.

A continuación, explicaremos qué funciones utiliza cada subapartado del *Main* para mostrar su información.

Los distintos agentes disponibles se muestran haciendo uso del archivo **ossec_lib_agent.php,** en concreto de la función *os_getagents($ossec_handle* , que nos devuelve los distintos agentes que tenemos configurados además de información adicional de estos. Esta información adicional consta de :

- Sistema operativo.
- Nombre.
- Tiempo desde el último cambio.
- Estado (conectado/no conectado).

En cuanto a los últimos archivos modificados, utilizan el archivo **ossec_lib_sys_check.php**, que tiene funciones que se encargan de parsear todos los "*syschecks*" de uno o todos los agentes. Estas funciones acceden al fichero */queue/syscheck* para sacar la información necesaria. Las principales funciones que se encargan de extraer la información relevantes de este fichero son:

- *os_getsyscheck($ossec_handle):* Nos devuelve una lista de los archivos modificados.

- *os_syscheck_dumpdb ($ossec_handle, $agent_name):* Hace lo mismo que la función anterior sólo que para un único agente, y volcando la base de datos de los hashes en la lista, pudiendo ver la diferencia de estos si se requiere.

Ambas funciones anteriores serán utilizadas además para las vistas "*Integrity Checking*" y "Stats". En el caso de la vista *Main* utiliza únicamente *os_getsyscheck($ossec_handle).*

Por último, se pueden ver los últimos eventos ocurridos, para mostrarlos se utiliza la función *os_getalerts()* del archivo **ossec_lib_alerts.php.**

- *os_getalerts($ossec_handle, $init_time, $final_time, $max_count)*: Devuelve una lista de alertas cumpliendo con los parámetros pasados. Accede al fichero "*logs/Alerts/alert.log*" para obtener los datos registrados de alertas.

Un ejemplo de uso de la función desde *Main* sería *os_getalerts($ossec_handle, 0, 0, 30)*, devolviendonos únicamente las últimas 30 alertas .

5.1.3 Search .php

Esta vista nos permite realizar búsquedas de alertas con distintos filtros, para poder ver así sólo las alertas que nos interesan en cada caso. Para poder buscar información y filtrarla, se utiliza la función *os_searchalerts()* del archivo **ossec_lib_alerts.php.**

- *os_searchalerts($ossec_handle, $search_id, $init_time, $final_time, $max_count, $min_level, $rule_id, $location_pattern, $str_pattern, $group_pattern, $srcip_pattern, $user_pattern, $log_pattern)*

Los parámetros que recibe coinciden con los filtros que se pueden aplicar en la búsqueda, que son los siguientes:

- *$search_id*: Dado que las búsquedas pueden guardarse, este parámetro marca el identificador con el que lo queremos nombrar.
- *$init_time*: Marca una fecha límite de inicio para mostrar sólo alertas posteriores a la fecha dada.
- *$final_time*: Marca una fecha límite final para mostrar sólo alertas anteriores a la fecha dada.
- *$max_count*: Número máximo de alertas que queremos mostrar.
- *$min_level*: Nivel mínimo de estas alertas.
- *$rule_id*: Buscar sólo alertas con un id determinado de regla.
- *$location_pattern*: Nos permite filtrar alertas por localización. Añadiendo antes a la búsqueda el carácter '!' significa la negación de la localización, es decir, mostrará alertas que no están en una determinada localización.

- *$str_pattern*: Utilizada para restringir las alertas por mensaje. Compatible con el carácter '!'.
- *$group_pattern*: Dejará filtrar la búsqueda por grupo de eventos. Estos grupos se encuentran definidos en **Ossec_categories.php** y son los siguientes:

    - Reconnaissance: Reconnaissance (all), Connection attempt, Web scan, Generic scan.
    - Authentication Control: Authentication Control (all), Authentication Success, Authentication Failure, Invalid login, Multiple auth failures, User account modified:adduser|account_changed, Policy changed.
    - Attack/Misuse: Attack/Misuse (all), Worm, Virus, Automatic attack, Exploit pattern, Invalid access, Spam, Multiple Spams, SQL Injection, Generic Attack, Rootkit detection.
    - Access Control: Access Control (all), Access denied, Access allowed, Invalid access, Firewall Drop, Multiple fw drops, Client mis-configuration, Client error.
    - Network Control: Network Control (all), New host detected, Possible ARP spoof.
    - System Monitor: System Monitor, Service start, Service in Risk, System error, Shutdown, Logs removed, Invalid request, Promiscuous mode detected, Configuration changed, Integrity Checking, File modification.
    - Policy Violation: Policy Violation (all), Login time violation, Login day violation.

- *$srcip_pattern*: Permite el filtrado de alertas por dirección IP de destino. Nuevamente compatible con el carácter '!'.
- *$user_pattern*: Filtro por usuario.
- *$log_pattern*: Restringe resultados por grupo de log. Estos grupos se encuentran definidos en **Ossec_formats.php** y son los siguientes:

    - Syslog: Syslog (all), Sshd, Arpwatch, Ftpd, Pam Unix, Proftpd, Pure-ftpd, Vsftpd, Sendmail, Postfix, Imapd, Vpopmail, Spamd, Horde IMP, Smbd, NFS, Xinetd, Kernel, Su, Cron, Sudo, PPTP, Named.
    - Firewall: Pix, Netscreen.
    - Microsoft: Microsoft (all), Windows, MS Ftp, Exchange.
    - Web Logs
    - Squid
    - Security Devices: Security devices (all), Cisco VPN, Symantec AV, NIDS.

5.1.4 Syscheck.php

Esta vista nos muestra un historial de los ficheros modificados, agrupados por días. Para llevar a cabo esta funcionalidad se hace uso de la función *os_getsyscheck($ossec_handle),* explicada anteriormente en el subapartado de ***Main.php***.

También desde esta vista podemos volcar la base de datos de las distintas funciones resumen, *hashes,* que estos archivos han ido teniendo a lo largo de su modificación, para un agente determinado, pudiendo así, ver los cambios. Esta funcionalidad se lleva a cabo mediante la función *os_syscheck_dumpdb ($ossec_handle,$ossec_aagent),* explicada anteriormente en la subsección 4.4.2.

5.1.5 Stats.php

Este fichero es el encargado de ofrecer a la WUI unas tablas de niveles de alertas.

Ossec_lib_stats.php: Realiza un parseo de todos los eventos que encuentre entre dos fechas proporcionadas, y los devuelve en forma de listado. El parseo se realiza sobre el fichero *stats/totals/date*. La búsqueda se realiza con la función *os_getstats($ossec_handle, $init_time, $final_time).*

5.2 Mejoras del Back

El código original de OSSEC no estaba diseñado para poder usar los datos recolectados de una manera eficiente si se quería añadir nuevas vistas, o hasta modificar las ya existentes.. Al no haber una clara diferenciación entre lo que llamaríamos vista, modelo y controlador (en una arquitectura MVC), hemos tenido que replantear la forma en la que recogemos los datos y los mostramos en la capa de visualización.

Muchas de las funciones mencionadas anteriormente devuelven directamente código HTML, por lo que decidimos separar la búsqueda de datos y procesamiento de los mismos, de la creación de la vista. Desde un archivo inicial (suele ir acompañado del nombre "***init***") para cada una de las vistas, hacemos peticiones a las funciones que encontramos de forma nativa en OSSEC. Si por cualquier razón los datos devueltos no forman una estructura de datos (una lista, un array, diccionario, etc.), cambiamos el código interno para que nos devuelva los datos dentro de una estructura de datos array, el cual luego usaremos dentro de cada vista.

Nuestro objetivo inicial nunca fue rehacer el código interno de OSSEC por completo, puesto que no tendríamos tiempo suficiente como para poder desarrollar una herramienta tan compleja desde cero. Nos hemos limitado a hacer ingeniería inversa para entender el funcionamiento del código, y de esa manera poder empezar a trabajar sobre él. Y a cambiar aquellas funcionalidades cuya mejora nos facilitaba el trabajo de desarrollo de un nuevo *frontend*.

En los siguientes subapartados explicamos los principales cambios que hemos realizado para cada una de las vistas.

5.2.1 Index / Main

En la versión original de OSSEC este archivo hacía de puente entre todas las vistas. Ahora, esta vista será nuestra vista Home. Para navegar por toda la página, en vez de tener una barra de búsqueda superior, o necesitar recoger un parámetro vía GET, hemos introducido un nuevo fichero llamado sidePanel.php que se encarga de referenciar la página que el usuario escoja.

Se han creado tres funciones, una por cada apartado que debe mostrar el front. Se ha creado la función *showAgents($ossec_handle)* dónde recupera todos los agentes existentes a través de la función *os_getagents($ossec_handle),* y recorre cada uno de ellos para mostrar su información, como podemos ver en *Código 5.2.1*.

```
function showAgents($ossec_handle){
 /* Getting all agents */
        if(($agent_list = os_getagents($ossec_handle)) == NULL)
        {
            echo "No agent available.\n";
            return(1);
        }
        /* Agent count for java script */
        $agent_count = 0;
        /* Looping all agents */
    foreach ($agent_list as $agent)
    {
        /* If agent is connected */
        if($agent{'connected'})
        {
            $atitle = "Agent active";
            $aclass = 'class="bluez"';
             $amsg = "";
        }
        else
        {
            $atitle = "Agent Inactive";
            $aclass = 'class="red"';
            $amsg = " - Inactive";
        }
        echo ' … ' /* Mostramos el html con la información */
```

Código 5.2.1

La segunda función desarrollada para esta vista es *showLastModified($ossec_handle), Código 5.2.2,* la cual se encarga de buscar los cinco últimos ficheros que han sido identificados por el syscheck

como modificados y mostrar la información de cada uno de ellos. Para lo cual utilizamos la función *os_getsyscheck($ossec_handle)* que nos devuelve un array de arrays, siendo uno de esos un listado de todos los ficheros modificados ordenados por fecha de última modificación *($syscheck_list{'global_list'}{'files'})*. Una vez obtenido los ficheros, se muestran con los atributos que deseamos mostrar al usuario.

```
function showLastModified($ossec_handle){
    $syscheck_list = os_getsyscheck($ossec_handle);
    if(($syscheck_list == NULL) || ($syscheck_list{'global_list'} == NULL))
    {
        echo '<ul class="ulsmall bluez">
            No integrity checking information available.<br />
            Nothing reported as changed.
            </ul>';
    }
    else
    {
       if(isset($syscheck_list{'global_list'}) &&
          isset($syscheck_list{'global_list'}{'files'}))
       {
           $sk_count = 0;
           foreach($syscheck_list{'global_list'}{'files'} as $syscheck)
           {
               $sk_count++;
               if($sk_count > ($agent_count +4))
               {
                   break;
               }
           $ffile_name = $syscheck[2];
           echo ' ... ' /* genera el html con la información */
```

Código 5.2.2

La última función desarrollada para esta vista es *listAlert($ossec_handle), Código 5.2.1,* que se encarga de mostrar las últimas alertas rescatadas a través de la función *os_getalerts($ossec_handle, $val1, $val2, $val3)*. Una vez recogidas las alertas, estas se muestran con toda su información.

```
function listAlert($ossec_handle){
    /* Getting last alerts */
    $alert_list = os_getalerts($ossec_handle, 0, 0, 30);
```

```
    if($alert_list == NULL)
    { echo "<b class='red'>Unable to retrieve alerts. </b><br />\n";}
    else{
        $alert_count = $alert_list->size()-1;
        $alert_array = $alert_list->alerts();

        while($alert_count >= 0)
        {
            echo '<div class="card shadow mb-4">
                <a href="#collapseCard'.$alert_count.'" class="d-block card-header py-3"
                    data-toggle="collapse" role="button" aria-expanded="false"
                    aria-controls="collapseCard'.$alert_count.'">
                    <h6 class="m-0 font-weight-bold text-primary">'
                    .$alert_array[$alert_count]->titleToHtml().'</h6>
                </a>
                <div class="collapse" id="collapseCard'.$alert_count.'"
                    <div class="card-body">';
                        echo $alert_array[$alert_count]->toHtml();
                echo'</div></div>';
            $alert_count--;
        }
    }
}
```

Código 5.2.3

5.2.2 Search

Para que la funcionalidad de búsqueda sea más organizada y atómica, hemos creado varios ficheros. El primer fichero es *search.php,* el cual muestra la totalidad de la vista. La información que muestra depende de algunas funciones como *os_searchalerts(...)* para rescatar las alertas según el filtro que introduzca el usuario.

Para el formulario creamos un fichero llamado *searchForm.php* que se encarga de recuperar las variables rellenadas por el usuario. Estas variables serán agrupadas en un array y se crea una opción con cada valor.

Adicionalmente, *search_variables.php* se encarga de recibir y comprobar todos los valores elegidos por el usuario e introducirlos a variables globales. Realizamos también la comprobación de que no se pueda introducir un valor erróneo para cada valor como por ejemplo una fecha final más antigua que la inicial, o valores numéricos en valores de tipo String entre otros. Podemos ver una parte del código en Código 5.2.4:

```
/* Getting location */
if(isset($_POST['locationpattern']))
{
    $lcpattern = "/^[0-9a-zA-Z.: _|^!>\/\\-]{1,156}$/";
    if(preg_match($lcpattern, $_POST['locationpattern']) == true)
    {
        $LOCATION_pattern = $_POST['locationpattern'];
        $u_location = $LOCATION_pattern;
    }
}
```

Código 5.2.4

Este código realiza una comprobación para que el valor de *location* únicamente contenga caracteres válidos, y en caso de que todos lo cumplan, entonces se introduce a la variable global. Una vez cambiados todos los valores, se vuelcan los datos en *search.php* dónde volvemos a ver el formulario y todas las alertas que hayan coincidido con los parámetros de búsqueda.

5.2.3 Integrity Checking

Esta funcionalidad nos permite comprobar si se ha modificado algún archivo comparando el hash almacenado con el actual, además de otras propiedades del fichero como su tamaño.

El primer inconveniente surge a la hora de recuperar las entradas en los archivos log al llamar a la función *os_getsyscheck($ossec_handle)*. Esta función está definida dentro del archivo *os_lib_syscheck.php*, y posteriormente llamará a *__os_getchanges(...)*. El funcionamiento de esta función consiste en generar un diccionario (en PHP los diccionarios se interpretan como listas o arrays anidados) que contiene toda la información que es posible recoger dentro de un buffer limitado a 1200 bytes.

El primer cambio realizado ha consistido en multiplicar por 10 el buffer por defecto, puesto que queremos poder hacer un análisis de muchos más datos desde una única llamada (en vez de en dos pasos como se hacía anteriormente). De esta manera podemos mediante una única llamada usarlos para la funcionalidad de volcado de la base de datos “Dump Database”. En el diseño original de esta funcionalidad por parte del equipo de OSSEC, cada entrada del array se corresponde con una entrada en el log, lo cual, a priori, nos impide poder hacer un agrupamiento por fecha u hora de modificación.. A continuación mostramos un fragmento del código original y el formato de los datos anterior (Código 5.2.5 y Código 5.2.6):

```
/* Global list */
if(sizeof($g_last_changes{'files'}) < 100)
```

```
{
          g_last_changes{'files'}[] =
                    ($time_stamp, $_name, $sk_file_name);


        if(!isset($g_last_changes{'lowest'}))
        {
                $g_last_changes{'lowest'} = $time_stamp;
        }
        if($time_stamp < $g_last_changes{'lowest'})
        {
                $g_last_changes{'lowest'} = $time_stamp;
        }
                rsort($g_last_changes{'files'});
}
else if($time_stamp > $g_last_changes{'lowest'})
{
        rsort($g_last_changes{'files'});
        array_pop($g_last_changes{'files'});

        $g_last_changes{'files'}[] =
                array($time_stamp, $_name, $sk_file_name);
}
```

Código 5.2.5

```
Array
{
          [files] => Array
          {
                    [0] => Array
                    {
                              [0] => 1591459624
                              [1] => ossec-server
                              [2] => etc/apparmor.d/usr.bin.firefox
                    }
                              [1] => Array
                                        { ...
```

Código 5.2.6

Los datos contenidos en el array aparecen como arrays anidados de N entradas (simplemente indexadas por valor incremental numérico), y dentro de cada uno de los arrays el "*timestamp*" (en segundos desde el inicio del sistema UNIX, el tiempo de modificación), el agente y el directorio del archivo. Si quisiéramos filtrar los datos por orden de antigüedad, o por nombre, tendríamos que recorrer una mayor cantidad de elementos (afecta al rendimiento), y no se podría hacer de forma intuitiva .

Reduciendo la cantidad de código, y modificando el formato de recolecta de datos, conseguimos un formato similar, pero más manejable.

```
if(preg_match($skpattern, $new_buffer, $regs)) {
          $current_time_formatted = date('Y-m-d', $regs[1]); // Current File
          if ($previous_file_time == "") {
                    $g_last_changes{'days'}[$dayCounter] = array($regs[1]); // Add new timestamp
                    $previous_file_time = date('Y-m-d', $regs[1]);
          } else if ($previous_file_time < $current_time_formatted) {
                    $dayCounter += 1;
                    $g_last_changes{'days'}[$dayCounter] = array($regs[1]); // Add new timestamp
                    $previous_file_time = date('Y-m-d', $regs[1]);
                    $filePerDay = 0;
          }
          $time_stamp = $regs[1]; // Seconds without format
         $sk_file_name = $regs[2]; // File Name
         // Fill in the day
         $g_last_changes{'days'}[$dayCounter]{'file'}[] = array($time_stamp, $_name, sk_file_name);
         $filePerDay += 1;
}
```

Código 5.2.7

```
Array
{
          [days] => Array
          {
                    [0] => Array
                    {
                              [0] => 1591459624
                              [File] => Array
                              {
                                        [0] => Array
                                        {
```

```
                                        [0] => 1591459624
                                        [1] => ossec-server ...
                                        [2] => /etc/apparmor.d/usr.bin.firefox
                                    }
                                    [1] => Array
                                    { ...
```

Código 5.2.8

Gracias a este formato, ahora se pueden aplicar una serie de filtros de forma atómica y precisa.

```
function applyFilters(&$array, &$filters, $dayCounter) {
    if ($filters['fileName'] != '') {
        $aux = [];
        for ($i = 0; $i < sizeof($array{'days'}); $i+=1) {
            for ($e = 0; $e < sizeof($array{'days'}[$i]{'file'}); $e++) {
                if (strpos($array{'days'}[$i]{'file'}[$e][2], $filters['fileName']) !== false) {
                    array_push($aux, $array{'days'}[$i]{'file'}[$e]);
                }
                /*if ($array{'days'}[$i]{'file'}[$e][2] == $filters['fileName']) {
                    array_push($aux, $array{'days'}[$i]{'file'}[$e]);
                }*/
            }
        }
        $array = $aux;

        if (sizeof($array) == 0) {
            $filters['fileName'] = 'ERROR';
        }
        return;
    }

    if ($filters['daySort'] == 'desc') {
        rsort($array{'days'});
    }
    if ($filters['fileSort'] == 'desc') {
        for ($i = 0; $i < sizeof($array{'days'}); $i+=1) {
            rsort($array{'days'}[$i]{'file'});
        }
    } ...
```

Código 5.2.9

Para unificar las funcionalidades que antes se encontraban separadas en dos pasos (comprobación de la integridad de los archivos y visualización de los directorios afectados), llamamos directamente a la función que nos recoge los datos de la base de datos, pero descartando los archivos que no se han visto comprometidos, ya que son de menor relevancia.

Al iniciar la vista, además de llamar a la función anterior, se llama también a *os_syscheck_dumpdb_custom($ossec_handle, $USER_agent, $filters)*. Hemos tomado la decisión de pasar los filtros por referencia ya que al terminar de recoger todos los dato, en caso de que hubiese algún error, se notificará más adelante sobreescribiendo el valor del filtro por el literal: *"ERROR"*.

El código original y el código nuevo en este caso no varían tanto en funcionalidad, sino en el formato en el que se recuperan los datos. Antes se formaba código HTML desde la función, que más adelante se verá mostrado en forma de tabla (lo cual no nos da mucho flexibilidad a la hora de moldear los datos a nuestro gusto). A continuación mostramos un fragmento del código original.

```
/* Printing db */
    echo '<br /><br /><table width="100%">';
    echo '
         <tr>
           <th>File name</th>
          ...
    foreach($db_list as $list_name => $list_val)
    {
        $sk_class = ">";
        $sk_point = "";

        if(($db_count % 2) == 0)
        {
            $sk_class = 'class="odd">';
        }

        if(isset($list_val{'ct'}))
        {
            $sk_point = '<a id="id_'.$list_val{'ct'}.'" />';
        }

        echo '<tr '.$sk_class.'<td width="45%" valign="top">'.
            $sk_point.
            $list_name.'</td><td width="53%" valign="top">'.
            $list_val{'sum'}.'</td>';
```

```
...
```

Código 5.2.10

Como podemos observar, se está dando formato e imprimiendo el código HTML directamente desde la función (mediante llamadas a la función *"echo"*), por lo que no permite modificar como se muestran los datos a no ser que se omitan estos pasos. El código nuevo tan solo genera la estructura de datos en la que irán metidos los datos:

```
foreach($db_list as $list_name => $list_val)
{
          $sk_class = ">";
          $sk_point = "";

          if(($db_count % 2) == 0)
          {
                              $sk_class = 'class="odd">';
          }

          if(isset($list_val{'ct'}))
          {
                              $sk_point = '<a id="id_'.$list_val{'ct'}.'" />';
          }
              $entry = array(
                              $db_count => array(
                         'name'=> $list_name,
                       'sum' => $list_val['sum'],
                         'size' => $list_val['size'],
                       'changed' => $list_val['changed']
                              )
                    );
          array_push($db_array, $entry);
          $db_count++;
}
```

Código 5.2.11

Ahora todos los datos que obtenemos serán volcados sobre un array (el cual tratamos como un diccionario) que será llevado al front para mostrarlo como se desee.

5.2.4 Stats

Como bien hemos mencionado en los apartados de análisis de la capa de front, la vista de *Stats* viene dividida en tres bloques. El primer bloque recoge las estadísticas basadas por el nivel de gravedad de la alerta, el segundo según el nombre de las reglas y el tercero es una estadística general ordenada por horas y días de las alertas.

Recogemos todas las estadísticas con la función *os_getstats($ossec_handle, $init_time, $final_time).* En la antigua versión, se mostraba las estadísticas ordenadas por el número de casos que existían en cada nivel. En la versión que hemos realizado en este trabajo hemos ordenado las estadísticas por el nivel concreto, ya que nos ha parecido más intuitivo.. Para implementarlo hemos creado un array donde el índice indica el nivel y el valor es el número de casos que existen en ese nivel. Todo esto se puede observar en el siguiente código.

```
$stats_list = os_getstats($ossec_handle, $init_time, $final_time);
$daily_stats = array();
if(isset($stats_list{$l_year_month}{$USER_day}))
{
    $daily_stats = $stats_list{$l_year_month}{$USER_day};
    $all_stats = $stats_list{$l_year_month};
}
if(!isset($daily_stats{'total'}))
{
    echo '<br />
        <b class="red">No stats available.</b>';
    return(1);
}
if( array_key_exists( 'level', $daily_stats ) ) {
foreach($daily_stats{'level'} as $l_level => $v_level)
{
    (int)$level_pct = (int)($v_level * 100)/$daily_stats{'alerts'};
    $val_array[$l_level]=$v_level;
}
```

Código 5.2.12

La parte del backend relacionado con las gráficas y las tablas está especificado dentro de las librerías propias de DataTables [15] y Google Charts [16] , por lo que no entramos en detalle en cuanto al funcionamiento de cada una de ellas. En el subapartado 4.2.3 se encuentra la explicación detallada de su uso.

5.2.5 Custom logs

Esta funcionalidad no estaba completamente desarrollada en la versión estable de OSSEC. La idea de introducirla nos surgió tras el análisis que realizamos de la documentación de OSSEC. El programa dispone de un script que sirve para registrar nuevas monitorizaciones.
A continuación, se muestra un pequeño análisis del script util.sh.Util.sh es un script para añadir ficheros, sitios webs o DNS para que sean monitorizados. Recibe como argumentos una opción y dependiendo de ésta, los atributos que son necesarios.
Si recibe la ruta de un fichero, realiza una búsqueda en *ossec.conf* para comprobar que no esté en la lista de archivos ya monitorizados y, si no lo está, añade el fichero con los atributos de localización de fichero (<*location*>) y el formato del fichero (<*log_format*>).
Si recibe un sitio web, realiza una búsqueda del dominio por lynx (navegador web) con el comando `lynx --connect_timeout 10 --dump $FILE`. Para que se pueda monitorizar, es necesario tener instalado este navegador. Si ha sido posible conectarse, este se añade en el ossec.conf (si ya no estaba ya monitorizando).
Al recibir un DNS, realiza una búsqueda de este para ver si existe con el comando `host -W 5 -t NS $FILE`. En caso de poder conectarse, revisa que no esté en el fichero de configuración, y lo añade para que pueda ser monitorizado.

Después de esta pequeña introducción, hablaremos sobre la creación de esta nueva vista y de la mejora del script. Si un usuario desea introducir una nueva monitorización, solo debe rellenar los datos en el formulario y estos valores serán almacenados en el fichero ossec.conf. El funcionamiento del back de esta vista es el siguiente:

Una vez recibido todos los valores del formulario se hace uso de ellos desde el fichero *customAdd.php.* Desde aquí, mediante llamadas a funciones, se va a encargar de crear archivos temporales para poder recoger todo el path completo del fichero y evitar que se origine una denegación de sistema de ficheros. El código que realiza esta función esta en Código 5.2.13

```
function tmpPath($ossec_tmp_path)
{
    $files = array();
    print_r($_FILES);
    //print_r($_FILES["location"]['error']);
    foreach ($_FILES["location"]["error"] as $clave => $error) {
        print_r($_FILES["location"]);
        if ($error == UPLOAD_ERR_OK) {

            $nombre_tmp = $_FILES["location"]["tmp_name"][$clave];
            $nombre = basename($_FILES["location"]["name"][$clave]);
            move_uploaded_file($nombre_tmp, $ossec_tmp_path.'/'.$nombre);
```

```
            array_push($files,$ossec_tmp_path.'/'.$nombre);
        }
    }
    return $files;
}
```

Código 5.2.13

Una vez terminada esta función, se realiza la llamada de ejecución del script *newUtil.sh*, que es el encargado de introducir la nueva configuración. Se puede observar en Código 5.2.14 de como se realiza:

```
if ($logFormat && $location && $check_diff && $only_future_events){
          shell_exec('./../sh/newUtil.sh addfile ' .$type_format.' '.$logFormat. ' ' .$location.
          ' ' .$check_diff. ' '.$only_future_events');
} else { throw new Exception;}
```

Código 5.2.14

El script cutomizado *newUtil.sh* tiene como base *util.sh*. Le hemos añadido la funcionalidad de añadir comandos mediante la función *addcommand*, añadiendo la posibilidad de añadir un fichero de tipo *shell scripting* para que sea monitorizado. Primero se comprueba si el archivo está siendo monitorizado o si ya existía previamente, igual que se realizaba en *util.sh* cuando queríamos añadir un un nuevo fichero. Una vez comprobado esto se realiza la introducción al *ossec.conf*. En la fragmento de código a continuación mostramos la condición incluir la respuesta activa a un comando.

```
if [ "X$DISABLE" = "X" ]; then
…
else
echo "
        <ossec_config>
           <command>
                <name>$NAME</name>
                 <executable>$FILE</executable>
                 <expect>$EXPECT</expect>
                 <timeout_allowed>$TIME</timeout_allowed>
           </command>
        <active-response>
            <disabled>$DISABLE</disabled>
            <command>$NAME</command>
            <location>$LOCATION</location>
```

```
            <level>$LEVEL</level>
            <rules_id>$RULES_ID</rules_id>
        </active-response>
        </ossec_config>
        ">> /var/ossec/etc/ossec.conf
    fi
```

Código 5.2.15

La otra mejora sobre este fichero ha sido la introducción de más atributos en los ficheros que se quieren monitorizar (*only-future-events* y *check_diff*), puesto que en el fichero *util.sh* solo existían los atributos *location* y *log_format* para esta opción. Además hemos añadido la posibilidad de monitorizar comandos ya existentes, ya que OSSEC ofrece la posibilidad de poder realizarlo si lo introducimos a *ossec.conf*. Para ello se debe ejecutar el siguiente script.

```
if [ $TYPEE = "command" ]; then
         echo "
            <ossec_config>
                <localfile>
                    <log_format>$LOGFORMAT</log_format>
                    <command>$COMMAND</command >
                    <alias>$ALIAS</alias>
                    <frequency>$FREQ</frequency>
                </localfile>
            </ossec_config>
        " >> /var/ossec/etc/ossec.conf
```

Código 5.2.16

6. División de tareas

En esta sección explicaremos cómo ha contribuido cada miembro del grupo al proyecto. No entraremos en demasiado detalle, debido a que en los apartados anteriores se ha explicado de forma más profundizada el desarrollo e investigación que ha realizado cada miembro.

6.1 Daniel Candil Vizcaíno

En la etapa de investigación, Daniel se encargó de obtener toda la información del funcionamiento, las características y componentes de OSSEC. Tras una gran búsqueda de información sobre OSSEC, acabó usando la documentación oficial por varios motivos: Primero, porque no da, a priori, datos erróneos. Y segundo, porque las demás búsquedas eran resúmenes de la documentación oficial [17].

Una vez buscada la información, Daniel realizó un pequeño estudio sobre OSSEC, donde se profundizó en cada componente que tenía la herramienta, para conocer los detalles de cada uno. Observó cómo funcionaba, qué procesos se utilizaban, como configuraba este HIDS la monitorización personalizada, como se realizan los test de reglas, ver ejemplos, etc. Y una vez analizado todo, realizó una pequeña documentación que fue subida al repositorio de Github, destacando lo más importante e interesante de OSSEC. Y con todo ello, lo expuso a sus compañeros para que tuviesen una comprensión más detallada de la herramienta.

En la etapa de desarrollo se encargó de varias tareas. La primera tarea fue introducir la pestaña *Main* en la nueva interfaz. La tarea que en principio iba a resultar fácil, se complicó debido a la complejidad del código original. Ésta se divide en tres partes: Agentes disponibles, ficheros modificados y últimos eventos. El código original no está dividido por funciones, ni clases, y eso dificultaba su comprensión. Daniel decidió entonces que, cada apartado, debería estar en su propio bloque, para que fuese más visible y comprensible. Cada nuevo bloque fue introducido a la plantilla de Bootstrap que elegimos, dejando más estructurado, más limpio y más sencillo el código, manteniendo la funcionalidad original de cada apartado.

Una vez terminada la tarea anterior, Daniel se ocupó de mostrar de forma gráfica las estadísticas de la alertas agrupadas por nivel de gravedad, dentro de la pestaña *Stats*. Daniel eliminó toda la parte del HTML del código, centrándose en la parte funcional, en este caso, cómo se recogían los datos. Una vez comprendido el código y habiendo realizado nuevas funciones, utilizó *Google Charts* [21] (en concreto las de tipo *Column Chart*), para mostrar los datos recogidos. Además cambió el orden que tenía originalmente (antes se ordenaba por el número de alertas, y ahora por número de nivel de alerta) y otorgando una selección de colores según la gravedad de la alerta, siendo el verde para alertas con menos gravedad y rojas las más graves. Estas nuevas características ayudan a la compresión de esta estadística.

Daniel tuvo la idea de incluir una nueva funcionalidad a la página web: “Custom Logs”. Investigando sobre OSSEC, Daniel encontró un fichero llamado *util.sh*. Este fichero servía para añadir nuevos registros para que fuesen monitorizados, pero de una forma muy concreta, ya que, sólo le otorgaba la posibilidad de especificar dos atributos: la ruta y el tipo de log. Además también existía la opción de poder monitorizar un DNS o un sitio web.

Para el desarrollo de la nueva funcionalidad, creó un nuevo fichero llamado *newUtil.sh*, basado en el anterior, donde incluyó la monitorización de DNS y sitio web. Aumentó el número de atributos que se pueden incluir a la hora de personalizar registros (los mismos que aparecen disponibles según la documentación). Adicionalmente, incorporó la monitorización de comandos, los cuales pueden ser de dos tipos: comandos ya existentes o comandos que pueden ser creados a través de un script. El primero a su vez, puede tener la opción de añadirle la respuesta activa con su respectiva configuración. Una vez introducidas las opciones de configuración, el fichero *newUtil.sh* las añade al fichero *ossec.conf*.

Una vez desarrollado el nuevo módulo, se añadió la pestaña *Custom Logs* al resto de vistas. En la vista encontramos un formulario en el cual se especifican los parámetros del nuevo registro a ser añadido. Junto al fichero PHP añadido, se programó en JavaScript un controlador de secciones para ir mostrando al usuario los atributos disponibles según lo que quiera monitorizar, ya que no son los mismos atributos para un comando que para una página web.

6.2 Guillermo Sánchez-Mariscal González

La investigación de Guillermo consistió en el análisis de la WUI de OSSEC. Para ello, lo primero que hizo fue instalar el software [18] que se referencia en la documentación oficial de OSSEC. Este software era útil para poder gestionar el HIDS desde el propio navegador, facilitando así su uso y la comprensión de sus alertas.

Después de instalar la WUI, quedaba pensar que información de la vista nos iba a ser de utilidad para poder desarrollar nuestro propio panel de gestión de alertas. Las distintas vistas que la WUI tenía eran útiles, porque reflejaban prácticamente todas las funcionalidades que la herramienta proporcionaba. El estudio de todas era necesario, y por lo tanto, requería investigar las funciones que éstas utilizaban y los ficheros de los que se extraía información de la monitorización. De todas las vistas, se encontraron funciones que nos iban a ser muy útiles. Simplemente cambiando la forma de extraer la información las íbamos a poder reutilizar en nuestra nueva interfaz.

Una vez extraída toda la información útil de la antigua WUI, Guillermo realizó documentación sobre el estudio, donde recogió toda la información adquirida en el análisis anterior, y así facilitar a sus compañeros la comprensión del código.

El desarrollo comenzó con la vista de *Search* . Esta vista era una de las más complicadas a priori, basándonos el estudio del código que se había realizado anteriormente. La dificultad de la vista, era poder filtrar las distintas alertas a través de los campos rellenados en el formulario. Esto se hace mediante expresiones regulares, que hemos podido reutilizar del código de la WUI anterior. Era necesario crear archivos temporales para poder guardar toda la información recogida por estos filtros, puesto que, su tamaño podía a llegar a colapsar la memoria. Poder crear estos archivos fue complicado porque no teníamos lo permisos necesarios desde la aplicación, pero ejecutándola como administrador el problema se resolvía. Además se añadieron gráficas con estadísticas de las alertas buscadas, donde podemos ver información como los distintos niveles de alertas, los distintos tipos y por último las IP origen de estas alertas.

La segunda tarea fue crear una tabla para poder visualizar las distintas reglas de acción o reglas personalizadas que se han detectado en una fecha determinada. Se utilizó DataTables para poder visualizar la información. La tabla permite poder exportar esa información en Excel, CSV y en PDF, y así poder generar documentación sobre estas reglas. Además permite ordenar las reglas por porcentaje de más activadas, incluso realizar un filtro por nombre o alerta.

Por último, Guillermo desarrolló la gestión de formularios del apartado de custom logs. La parte que llevó más tiempo desarrollar fue el drag and drop de los archivos que queríamos monitorizar, puesto que, para poder ejecutar el newUtil.sh necesitamos la ruta completa del lfichero.

6.3 Adrián Ruiz Householder

En la etapa de investigación, Adrián ha sido el encargado de buscar librerías y plantillas para el frontend de la nueva WUI de Ossec. El reto principal era encontrar una librería que, además de ser open-source, fuese modificable [19] y se basara en Bootstrap [20], ya que es una biblioteca con la que hemos trabajado anteriormente.

Una vez elegida la librería más adecuada, Adrián hizo un duplicado de la WUI antigua y tradujo de manera provisional todos los componentes HTML, así como la organización de directorios y librerías. Se añadieron algunas vistas, como la de Error 404, y se creó una plantilla para reutilizar en futuras vistas. Durante el desarrollo del proyecto, Adrián ha ido mejorando y actualizando la capa visual de todas las vistas.

Adrián se ha encargado de mejorar la vista de *Integrity Checking*. En esta vista, encontramos un desglose por días de los archivos que han sido modificados, acompañado de la hora de modificación. El principal problema, aparte de que fuese poco intuitivo, es que todos los datos aparecían sin formato. Esto para un usuario puede resultar bastante abrumador, puesto que la interfaz no es autoexplicativa.

El primer cambio, fue simplemente visual: convertir cada registro en un desplegable, de forma parecida a como estaba antes. Con esto, surgieron otra serie de ideas para que fuese más interactivo aún. Se añadió la posibilidad de agrupar los registros por día, y luego por mes, de forma que hubiesen varios desplegables anidados.

Un problema que tenía esta vista en la aplicación original es que la funcionalidad de verificación de integridad (accesible después de hacer click en el botón "Dump Database") se encontraba separada de la visualización del historial de archivos modificados. Se unificaron estas funcionalidades para que el usuario no tuviese que fijarse en tantos datos a la vez, y pueda hacerlo todo de una manera organizada.

Más adelante se añadió la posibilidad de filtrar los datos por orden de modificación (creciente - decreciente), limitar el número de archivos que se muestren y hacer una búsqueda por nombre o por hash de los archivos que se recuperan. Esto permite que se pueda comprobar por completo la integridad de un archivo en concreto, sin necesidad de tener que ver todo el historial.

En la vista *Stats*, Adrián fue el encargado de fusionar las partes de cada uno, además de programar las últimas dos gráficas, que muestra el historial por hora y por mes de las alertas almacenadas. Cada una de las gráficas utilizadas parten de las librerías de Google Charts [21] ya que se pueden utilizar de forma gratuita y son muy versátiles, además de ya estar familiarizado con ellas.

Antes de comenzar el desarrollo de la WUI, se puso en contacto con los desarrolladores de Ossec, para explicarles en qué consistía nuestro proyecto. Nos respondieron y nos invitaron a un grupo de Slack para preguntarles cualquier cosa que hiciera falta.

7. Herramientas

En este apartado describiremos las tecnologías que hemos utilizado durante el desarrollo de nuestro proyecto: el uso de virtualización, el entorno de desarrollo donde hemos programado, el uso de control de versiones, los lenguajes y el servidor web manejado

7.1 Virtualización:

- **VirtualBox**: Virtual Box es un software de virtualización. Nos permite alojar un sistema operativo en otro anfitrión. En nuestro caso hemos utilizado las versiones 5.2.6 y 6.1.6, instalado en Windows 10. La imagen utilizada es Ubuntu.
- **Imagen Ubuntu:** Ubuntu es un sistema operativo distribuida por Linux de software libre y código abierto. La instalación de OSSEC y su interfaz web de usuario (WUI) para el desarrollo de nuestro trabajo se realizó en una imagen Ubuntu, versión 18.04 alojado en VirtualBox.

7.2 Entornos de desarrollo:

- **Visual Studio Code**: El IDE que hemos utilizado durante el desarrollo de todo el proyecto ha sido Visual Studio Code, que permite personalizar por completo el entorno y añadir plugins para facilitar el trabajo. Para nuestra ayuda hemos utilizado el siguiente plugin:
    - Live Share: es un plugin que permite programar en grupo en tiempo real. Un usuario comienza un sesión (servidor) a la que se pueden unir más personas mediante invitación, pudiendo editar un mismo proyecto a la vez. Además, permite compartir puertos, lo cual permite que podamos acceder al *localhost* de otra máquina en remoto.

7.3 Control de versiones:

- **Github**: es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Nuestro proyecto ha sido alojado en esta plataforma, la cual nos ha servido para poder trabajar con el código actualizado, en nuestra propia rama y poder llevar un control sobre el. Para ello, hemos utilizado las herramientas Tortoise y Git Bash para gestionar los archivos almacenados en el repositorio local. Complementariamente, hemos utilizado Github para alojar ciertos manuales y explicaciones que hemos realizado sobre OSSEC. Esto nos ha ayudado a la hora de recoger y rememorar información de este HIDS

- **Tortoise SVN:** Para acceder al repositorio de forma rápida y más gráfica uno de los miembros del equipo decidió usar esta aplicación puesto que ya había trabajado con ella anteriormente. Es un cliente de *Subversion* [22] implementado como una extensión de la Shell de Windows, que ejecuta los comandos sin que tengamos que meterlos a mano. Uno de los puntos positivos, aparte de ser gratuita, es que añade los comandos de control de versiones a la lista de opciones sobre un directorio o archivo al hacer click derecho sobre él. Desde ahí, podemos hacer "checkout" del repositorio, y desde ese momento, los "commit" y "update", que hacen referencia a los comandos de "pull" y "push" sobre una rama, se hacen de forma rápida e intuitiva. Durante el proceso de cualquiera de estos dos comandos, podemos añadir un comentario y elegir los archivos que queremos subir al repositorio, mostrando un log con los archivos que han sido modificados.

7.4 Lenguajes:

- **PHP:** Es un lenguaje de código abierto especialmente adecuado para el desarrollo web y que permite que se intercale con HTML. Hemos utilizado este lenguaje para el back por diversas razones: nos resulta un lenguaje cómodo, ya que todos lo conocíamos de antemano, y la segunda razón viene causada por la WUI que tenía implementada OSSEC, ya que la mayor parte estaba realizada en este lenguaje y nos facilitaba reutilizar parte del código ya implementado.

- **HTML:** Es un lenguaje markup usado para estructurar y presentar el contenido de la web. Con HTML hemos realizado la estructura de nuestra página web, para que tenga un buen formato, limpio y sencillo de entender.

- **CSS**: Es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Hemos hecho uso de las librerías de Bootstrap, de las que hablaremos más adelante.

- **JavaScript:** Es un lenguaje de programación orientado a objetos, el cual hemos usado para programar todo tipo de funcionalidades añadidas a la capa de front de la nueva WUI.

- **Shell**: Es un intérprete diseñado para ejecutar scripts en Unix. Las operaciones que se suelen ejecutar incluyen manipulación de archivos, ejecución de programas e impresión de texto. La shell que hemos utilizado es Bourne Shell (.sh) cuya función es interpretar órdenes. La hemos utilizado para la creación del fichero *newUtil.sh* el cual consigue almacenar en ossec.conf una nueva monitorización personalizada.

7.5 Diseño

- **Bootstrap**: Es un framework que ayuda a diseñar aplicaciones web de una forma más fácil y sencilla. Incluye plantillas de diseño basadas en HTML, CSS y da soporte para complementos de JavaScript. En nuestra nueva interfaz hemos usado como base la plantilla *SB Admin 2* [20]. La hemos modificado y añadido estilos, y configurado nuestra estructura HTML, dejando una interfaz más intuitiva y organizada, y con un código más limpio y ordenado.

- **Google charts**: Es un servicio web interactivo que crea gráficos a partir de información proporcionada por el usuario. Google charts aparece en la pestaña de estadísticas de nuestra web, donde muestra al usuario de forma gráfica toda la información que desea ser rescatada y mostrada. Esta herramienta hace que para el usuario sea más fácil comprender y juzgar los resultados.

7.6 Servidor web

- **Apache**: Es un servidor web HTTP gratuito y de código abierto bajo la licencia Apache 2.0, siendo uno de los servidores web más utilizados a nivel mundial. Una de las utilidades que tiene Apache, y de la que nos hemos aprovechado, es la visualización local y probar el código durante su desarrollo.

8. Conclusiones (ESP)

La herramienta seleccionada, OSSEC otorga al sistema una capa adicional de seguridad bastante robusta, y permite personalizar su configuración para un mayor control. Aquí presentamos las conclusiones del trabajo desarrollado. En los siguientes subapartados queremos exponer alguna idea que tenemos sobre el futuro de nuestro trabajo, así como mencionar algunas de las asignaturas que más nos hayan ayudado a la hora de desarrollar algo como lo que hemos desarrollado en este proyecto.

OSSEC en definitiva ha resultado ser una herramienta muy potente ya que tiene una capa de seguridad configurable. OSSEC no sólo monitoriza, además consigue obtener unas estadísticas que nos facilita la visión de las alertas y una rápida comparación general de estas anomalías. La búsqueda de alertas y de integridad de ficheros, logra que el usuario sea más ágil en la exploración de intrusiones y modificaciones de archivos, dando un tiempo valioso a la hora de aplicar la seguridad necesaria.

El trabajo de ingeniería inversa desarrollado para entender no sólo las funcionalidades si no también cómo estaba implementado el código. Ha supuesto un tercio de todo el trabajo realizado en este proyecto pero nos ha ayudado a entender el funcionamiento de esta herramienta, comprender los beneficios que supone el instalar un HIDS en nuestro equipo y nos ha permitido expandir su funcionalidad para hacerla aún más sólida.

Hemos conseguido rediseñar una herramienta de forma que cualquier usuario que actualmente trabaje con dashboards similares, pueda fácilmente dar uso a todas las funcionalidades de OSSEC. Independientemente de todos los procesos que hay detrás de un software, sin una buena representación gráfica sus resultados, no se consigue realmente demostrar su potencial. Actualmente, se desarrollan muchas aplicaciones que podrían triunfar, pero que carecen de un buen soporte, o de un buen diseño gráfico. Nosotros hemos conseguido hacer que una herramienta considerada “antigua”, con un backend formidable, pueda volver a ser considerada novedosa, tan solo atendiendo a la apariencia de la aplicación.

Dicho esto, nuestro trabajo no se ha limitado sólo a desarrollar una nueva interfaz gráfica. El proceso de desarrollo no ha sido lineal, y en muchas ocasiones, hemos tenido que reestructurar el código original para satisfacer nuestras necesidades. En general, al rediseñar una aplicación, como hemos hecho nosotros, no basta con trasladar el funcionamiento a una nueva capa visual. Muchas veces no se piensa en el futuro, ya que la vida útil de una aplicación puede variar. Esto hace que se tomen decisiones de codificación que puedan no ser fácilmente trasladadas a una nueva implementación. La metodología que hemos utilizado a la hora de desarrollar el proyecto, se ha centrado en que todo el código realizado sea fácilmente reutilizado. Bootstrap, como muchos otros frameworks, buscan estandarizar el diseño gráfico para este tipo de situaciones, siempre y cuando el código backend se vea claramente diferenciado del frontend, tarea que hemos tenido que realizar en este trabajo. Ésta última parte del trabajo ha hecho que tengamos un conocimiento más completo de cómo funciona OSSEC.

Como conclusión, el desarrollo de esta herramienta nos ha ayudado a comprender cómo funciona internamente un HIDS. Hemos aprendido cómo es su estructura, que archivos monitoriza, cómo previene ataques y otras muchas funcionalidades de OSSEC. Todo este conocimiento nos permitiría poder desarrollar en un futuro nuestro propio HIDS, de una manera mucho más rápida y pudiendo aplicar nuevas funcionalidades que no tengan otros HIDS. Además, haber desarrollado una WUI, nos ha hecho ser conscientes que muchas aplicaciones pueden quedarse obsoletas a pesar de tener un backend potente, y por lo tanto, es tan importante tener desarrollado un backend robusto como un frontend que facilite a los usuarios sacar el máximo partido a la herramienta. Consideramos que en este trabajo este último punto lo hemos conseguido, nuestro frontend proporciona un valor añadido a OSSEC.

8.1 Futuro

La mejora y modernización del entorno web de OSSEC ha sido al final la parte del trabajo que nos ha llevado más tiempo y esfuerzo, además, consideramos que tener un entorno como el generado era esencial para poder trabajar adecuadamente con OSSEC. Existen diversas funcionalidades que nos habría gustado añadir para mejorar claramente la herramienta OSSEC. Por ejemplo, introducir más tipologías de configuración en el apartado de "*Custom Logs*" de la web, ya que ahora solo permite introducir ficheros, comandos, sitios web y DNS para su monitorización. Querríamos además añadir la posibilidad de poder introducir toda la configuración manualmente, accediendo

directamente a los archivos sin necesidad de rellenar un formulario, ya que a muchos administradores de sistemas les resulta cómodo y están acostumbrados a ello.

Por otro lado, nos gustaría conseguir implementar el aprendizaje automatizado, como mencionamos al inicio de las motivaciones. Una vez organizado el código, introducir esta funcionalidad sería uno de los siguientes pasos a realizar. El aprendizaje automatizado ayudaría a la herramienta a encontrar futuras intrusiones y añadir instantáneamente los ficheros a analizar sin que lo tenga que realizar el usuario. Esto también permitiría que tengamos una herramienta actualizada, algo importante en cuanto a la seguridad, ya que continuamente aparecen nuevos tipos de ataques y cada vez más sofisticados.

Por último, creemos que habría sido interesante añadir una comparativa entre agentes disponibles en una vista unificada, ya que puede resultar provechoso la equiparación de resultados. Con esto permitiremos al usuario analizar los agentes de una forma más global, saber qué host es más atacado y cuál necesita más blindaje para estar más protegido.

Si se diese el caso de que algún equipo quisiera, a partir de nuestro trabajo, desarrollar más funcionalidades para OSSEC, consideramos que lo tendrían más fácil, ya que nuestra interfaz es más amigable. Un ejemplo sería añadir un sistema de autenticación de usuarios del sistema, de forma que se puedan otorgar ciertos privilegios únicamente a aquellos usuarios que lo tengan permitido. Crear una vista de gestión de usuarios con un formato parecido a cualquiera de las vistas ya existentes, usándolas como plantilla, sería bastante sencillo.

8.2 Asignaturas relacionadas

En este bloque hablaremos de las asignaturas de nuestro grado que más relación tienen el trabajo realizado.

En primer lugar destacaremos la asignatura de *Redes y Seguridad*, donde estudiamos de manera básica que era un IDS, su funcionamiento y la importancia que tienen para detectar intrusiones no deseadas. Además en esta asignatura vimos los principios de la seguridad, que hemos ido aplicando a lo largo del desarrollo de nuestro trabajo.

La asignatura de *Aplicaciones Web* nos ha ayudado a la obtención de conocimientos básicos y necesarios para el desarrollo de nuestra interfaz de una manera más sólida y consistente. Es en esta asignatura donde comenzamos el aprendizaje de los lenguajes de HTML, CSS, JavaScript y PHP, justamente los lenguajes que hemos utilizado.

Por último, *Sistemas Operativos* nos aportó el entendimiento de los procesos y los daemons, necesarios para interpretar el funcionamiento de OSSEC. También para la ejecución de comandos y el lenguaje shell para la funcionalidad de "custom log".

Estas han sido las asignaturas que más relevancia han tenido hacia nuestro proyecto, pero seguramente podríamos mencionar otras que gracias a su realización han hecho más fácil el desarrollo de este trabajo.

Todo el código mencionado a lo largo del proyecto puede ser encontrado en GitHub [23].

8. Conclusions (EN)

OSSEC, the tool we've chosen, adds a very robust security layer to any system, allowing the user to modify its configuration for a better control. Here we estate our final conclusions to the work we have done. In the following sections, we will talk about the future of our project, as well as mention some of the subjects that have had the most impact on our work.

In the end, OSSEC has turned out to be a very powerful tool since it allows us to modify its security layer. Not only does OSSEC monitor the system, but it also gathers statistics which help understand the significance behind alerts and to quickly draw comparisons between them. Searching for alerts and checking the integrity of files allows the user to quickly browse through the most relevant intrusions and modifications of files, ultimately increasing the time available to apply any necessary changes to the security protocols.

Reverse engineering has helped us better understand the implementation of the code as well as the software's functionality.Even though it has taken a third of the total of the time we have spent on the project, it has resulted in the better understanding of the project as a whole, as well as the many benefits there are in installing an HIDS in our system, as well as further expand its functionalities to make it even more solid.

We have managed to redesign a tool so as to easily allow any user, who is currently working through a similar dashboard, to take advantage of all the functionalities OSSEC has to offer. Regardless of all the processes behind a given software, without a good visual representation of their results, it is not possible to fully demonstrate its potential. Currently, many applications perfectly fit for success are developed, but lack good design decisions and support. We have made it possible to modify such software, considered "old", that has such a solid backend code, ultimately making it seem updated and modern by just changing the visual aspect of it.

Having said that, our work has not just been remodeling the graphical interface. The development process has not been linear, since, in multiple occasions, we have needed to restructure the original code to make our work easier. In general, when redesigning an application, which we have done, it is not enough to directly translate its current functionalities to the new graphical user interface. The vast majority of times the future of an application is not taken into account, since the lifespan of any given software may vary. This results in making decisions based on the current job at hand, without considering the future to come. Our methodology during this process has been based on the idea of allowing the code to be reused. Bootstrap, as well as many other frameworks, tries to standardize graphic user interfaces to deal with such issues, but only if the backend code is clearly separated from the frontend, a task we have worked on during the development of this project. This last part has given us the chance to better understand the inner workings of OSSEC.

In conclusion, the development of this software has helped us understand how a HIDS functions. We have learned how it is structured, the files it monitors, how attacks are prevented and many other of OSSEC's functionalities. All of this information could allow us to develop in the future a HIDS which incorporates functionalities that other HIDS do not. Furthermore, having developed a WUI we are now more aware that any application can be seen as obsolete regardless of how good the backend code is, therefore both the backend and the frontend must be robust enough to allow the users to fully utilize the tool. We consider this to be a goal we have been able to achieve, hence adding more value to OSSEC.

8.1 Future

The renovation and modernization of OSSEC's web user interface has been the most tedious task, which we considered essential in order to correctly work with this software. However, we would have liked to add more functionalities to its core. The first improvement would be adding more configuration topologies in our "*Custom Logs*" section, since it currently only allows us to register new commands, websites, files and DNS. Another key functionality that we would like to add is the possibility to manually modify the configuration files without having to fill out a form, since it is very comfortable for a lot of system administrators who are accustomed to such processes.

We would also like to incorporate machine learning, as we stated previously in our motivations. Once the code has been restructured, adding this feature would be the next step. Machine learning would help the tool find unknown intrusions and instantly register files to monitor without having to manually do so. This would result in making the software self-updatable and up-to-date, which is key when it comes to security, since every single day newer and stronger attacks arise.

Furthermore, it would be interesting to add a unified agent comparison function so as to visually understand how different each monitored system is working. This would allow the user to analyze every single agent in order to detect which of the hosts are more vulnerable, which are more prone to be attacked and which need more protection.

If any group of developers in the future would like to further add functionalities to OSSEC, using our project as a starting point, we consider it to be an easier task now that the WUI is more user-friendly. An example would be adding a user authentication system, that only allows certain users to be granted access to the functionalities of the application. Developing a user-management tab would be a fairly easy task, using any of the views that are currently finished as a template.

8.2 Related Subjects

In this section we are going to briefly mention some of the subjects we feel have had more impact during the development of our project.

First of all we would like to talk about *Security and Networks*, where we learned the basics of IDS, how they work and their importance in maintaining intruders aways from our system. Moreover, we learned the principles of security, which we had to take into account while working on our project.

Web Applications has taught us the basics of developing a solid and consistent user interface. Here is where we first learned to work with HTML, CSS, JavaScript and PHP, languages that we have been constantly working with during the revamp of OSSEC's WUI and backend.

Last but not least, *Operating Systems* has allowed us to understand the processes behind an operating system, as well as daemons, which are necessary to analyze OSSEC's inner workings. Everything related to the execution of commands and scripting has helped us develop our "Custom Logs" module.

All of the subjects we have mentioned above have had the most impact over the development of our project, but there are many others which we owe the level of knowledge we have today.

All of the code mentioned above can be found on GitHub [23].

9. Referencias

[1] Eckmann, Steven & Vigna, Giovanni & Kemmerer, Richard. (2001). STATL Definition. https://www.researchgate.net/publication/2379514_STATL_Definition

[2] Low, Wai & Lee, Joseph & Teoh, Peter. (2002). DIDAFIT: Detecting Intrusions in Databases Through Fingerprinting Transactions.. 121-128. https://www.researchgate.net/publication/220710728_DIDAFIT_Detecting_Intrusions_in_Databases_Through_Fingerprinting_Transactions

[3] Andreas Müller. (2009). VDM — The Vienna Development Method http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.153.3064&rep=rep1&type=pdf

[4] OSSEC About. (2020), OSSEC Project Team. https://www.ossec.net/about/

[5] OSSEC Documentation. Manual: Log monitoring/analysis: Configuration Options. 2010-2020, OSSEC Project Team. https://www.ossec.net/docs/docs/manual/monitoring/index.html

[6] OSSEC Documentation. Reference: ossec.conf: Syscheck Options. 2010-2020, OSSEC Project Team. https://www.ossec.net/docs/docs/syntax/head_ossec_config.syscheck.html

[7] OSSEC Documentation. Reference: ossec.conf: Rootcheck options. 2010-2020, OSSEC Project Team. https://www.ossec.net/docs/docs/syntax/head_ossec_config.rootcheck.html

[8] OSSEC Documentation. Reference: Log Analysis Syntax: Decoders Syntax. 2010-2020, OSSEC Project Team. https://www.ossec.net/docs/docs/syntax/head_decoders.html

[9] OSSEC Documentation. Reference: Log Analysis Syntax: Rules Syntax. 2010-2020, OSSEC Project Team https://www.ossec.net/docs/docs/syntax/head_rules.html

[10] OSSEC Documentation. Reference: ossec.conf: Active Response Options: Command Options. 2010-2020, OSSEC Project Team. https://www.ossec.net/docs/docs/syntax/head_ossec_config.active-response.html#command-options

[11] OSSEC Documentation. Reference: ossec.conf: Active Response Options: Active-response Options. 2010-2020, OSSEC Project Team. https://www.ossec.net/docs/docs/syntax/head_ossec_config.active-response.html#active-response-options

[12] OSSEC Documentation. Manual: Output and Alert options: Sending output to a Database. 2010-2020, OSSEC Project Team. https://www.ossec.net/docs/docs/manual/output/database-output.html

[13] OSSEC Documentation. Manual: Output and Alert options: Sending output to prelude. 2010-2020, OSSEC Project Team. https://www.ossec.net/docs/docs/manual/output/prelude-output.html

[14] Github: Blackrock Digital: startbootstrap-sb-admin-2. 2019, Your Website. https://blackrockdigital.github.io/startbootstrap-sb-admin-2/

[15] Google charts: Guides. 2019, Google https://developers.google.com/chart/interactive/docs
[16] DataTables. 2007-2020, SpryMedia Ltd. https://datatables.net/
[17] OSSEC Documentation. 2010-2020, OSSEC Project Team. https://www.ossec.net/docs/
[18] Github: OSSEC: Ossec-wui. 2006-2013 Trend Micro Inc.
https://github.com/ossec/ossec-wui.git
[19] Bootstrap. Build fast, responsive sites with Bootstrap (2011-2020): https://getbootstrap.com/
[20] BlackrockDigital SB-Admin 2: https://startbootstrap.com/previews/sb-admin-2/
[21] Google Charts (2008-2020): https://developers.google.com/chart
[22] Subversion (software). Apache SVN: https://es.wikipedia.org/wiki/Subversion_(software)
[23] GitHub New Enhanced OSSEC WUI: https://github.com/Mariscal6/hids_www